



Computing and Visualizing Concept Lattices

Dissertationsschrift

genehmigt vom Fachbereich Informatik
der Technischen Universität Darmstadt

von

Master des Wissenschaft Serhiy Yevtushenko

aus Kyiv, Ukraine

zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaft (Dr. rer. nat.)

Darmstadt 2004

Hochschulkennziffer D 17

Erstreferent: Prof. Dr. Wolfgang Bibel

Koreferent: Prof. Dr. Bernhard Ganter

Tag der Einreichung: 29. Juli 2004

Tag der Disputation: 7. Oktober 2004

Declaration

I hereby declare that this submission comprises my original work except where due acknowledgement has been made in the text to all other material used.

Signed:

To the memory of my Mother

Acknowledgements

First of all, I would like to acknowledge the constant support and the incredible patience of my thesis supervisor, Prof. Wolfgang Bibel. I am really grateful that I have the opportunity to work at the Intellectics group.

I would like to thank a lot to the current and former members of the Intellectics group for many interesting discussions, their help and a lot of useful advice and just creating environment in which it is fun to work. In particular I would like to thank Thomas Stützle, Gunter Grieser, Hesham Khalil, late Hartmut Bittner, Marco Chiarandini, Ulrich Scholz, Luis Paquete, Marco Pranzo, Tommaso Schiavinotto, Irina Dumitrescu, Mauro Birattari and Klaus Varrentrapp.

I am indebted to Peter Grigoriev for a friendship, helping me to adapt in Germany, a joint work that was jolly to do and lot of fun minutes.

I would like to thank Prof. Bernhard Ganter for inviting me several times to Dresden and interesting and helpful discussions.

I wish to express my gratitude to Prof. Tatyana Taran for guiding my first research attempts.

I am grateful to Sergei Objedkov and Sergei Kuznetsov for scientific cooperation and a lot of helpful suggestions and discussions.

I am grateful to Nataliya Ivashchenko for being a friend and her support, help, encouragement and giving me inspiration to start writing the thesis. And last, but not least, I'd like to thank Elena Kravchenko for being a real friend and providing support in the hardest times.

Contents

1	Introduction	1
2	Formal Concept Analysis	5
2.1	A short introduction	5
2.2	Basic terminology	5
2.3	Line diagrams of concept lattices	8
2.4	Implication sets	11
2.5	Relationship between three kinds of knowledge representation in FCA	13
2.6	FCA and knowledge discovery in databases	14
2.7	FCA from the viewpoint of KR	16
2.7.1	Mapping between terminology of FCA and DL	18
3	FCA Algorithms	19
3.1	Usage scenarios	19
3.2	Desirable properties of algorithms	21
3.3	Construction of the set of concepts	22
3.4	Decomposition of context w.r.t. an attribute	29
3.5	Computational complexity of algorithms	35
3.5.1	General intractability of the task of computing all con- cepts	35
3.5.2	Worst case complexity of algorithm from Fig. 3.3 . . .	36
3.5.3	Worst case complexity of algorithm from Fig. 3.6 . . .	37
3.6	Experimental comparison of the algorithms	37
3.6.1	The methodology of comparison	37
3.6.2	The results of the comparison for artificial datasets . .	39
3.6.3	The results of the comparison on the real-world datasets	40
3.7	Further ways of improving the algorithm	43
3.8	Using implicit representations	44
3.8.1	Binary Decision Diagrams	45
3.8.2	Using a BDD to represent the set of all intents	46

3.8.3	Calculation of all intersections between two sets	49
3.8.4	The algorithms using BDD for the construction of the set of all intents	49
3.8.5	Experimental comparison	51
3.8.6	Conclusions	54
3.9	Related work	55
3.9.1	Batch and incremental algorithms	56
3.9.2	Batch algorithm	61
3.9.3	Data mining-based algorithms	63
3.9.4	Other methods	67
3.10	Conclusions	67
4	Search Space Analysis	71
4.1	Characteristics of rule mining datasets	71
4.2	The distribution characteristics of lattices	74
4.3	Exploration of binary datasets for JSM	79
4.3.1	Short information about the JSM-method	79
4.3.2	Description of experiments	82
4.4	Effects of context transformations	82
4.5	Conclusions	86
5	Visualizing Concept Lattices	89
5.1	General requirements to drawing line diagrams	89
5.2	The layered approach	90
5.3	The force-directed approach	91
5.3.1	Force-directed scheme	92
5.3.2	Freese method	93
5.4	Additive line diagrams	94
5.4.1	Embedding of the lattice into n-dimensional grid	95
5.5	Comparison of methods	95
5.5.1	Test lattices	95
5.5.2	Results	96
5.6	Related work	100
5.7	Conclusions and future work	101
6	Conclusions	103
6.1	Contributions	103
6.2	Future work	105

Chapter 1

Introduction

One of the tasks of Artificial Intelligence is to model abilities that are generally considered as human by means of computers. One such ability is to analyse data and make decisions on the basis of the results. Another ability that is tightly connected with the first one is to represent knowledge and perform reasoning on the basis of this knowledge.

Among the areas of the artificial intelligence in which these abilities are explored and used in applications are Machine Learning and related to it Knowledge Discovery in Databases (KDD), sometimes also referred as a Data Mining. In recent years the number of applications of the Data Mining has swiftly grown. It is being widely applied in such diverse areas as market basket analysis, analysis of dependencies in biological sequences, search and extraction of information in the web and many others.

One of the methods of data analysis that is gaining a recognition is Formal Concept Analysis (FCA). This method can be nicknamed as “applied lattice theory” due to its roots in the community of researchers in lattice theory. It was mainly developed by the members of the Darmstadt group on Formal Concept Analysis.

The peculiarity that distinguishes FCA from many other analysis methods is the absence of a loss of information during the analysis of data. This peculiarity is at the same time an advantage and a disadvantage of the method: it is an advantage because the user can be sure that no important details were left out, and a disadvantage because the computational expenses arising when applying this method are very high.

The FCA unites the usage of three representations, namely, contexts (binary relations), concept lattices and implications for the analysis of data.

The formal context can be viewed as an object-attribute table, in which, if object o has attribute a , there is a cross in the cell on the intersection of the row corresponding to object o and the column corresponding to attribute

a. The maximally filled rectangular subtables of the context table with row set A and column set B correspond to the formal concept (A, B) . The set of objects A is called the extent and the set of attributes B is called the intent of the formal concept (A, B) .

Between concepts of a context a natural superconcept-subconcept relationship is established. The set of all concepts of the context forms a complete lattice called concept lattice. Usually a concept lattice is visualized by means of the Hasse (or line) diagram. The examination of the concept lattice by the expert allows to reveal dependencies which exist in the data.

The task of computing the set of all concepts and its line diagram are of central importance for applications using FCA. Also, it plays an important role in several fields of data analysis that are related to Formal Concept Analysis, namely, the associations analysis and the JSM method for hypothesis generation. The number and the scale of the application of FCA based techniques has grown considerable in the recent time. Because most algorithms proposed for the Formal Concept Analysis were compared and tested only on small datasets, in this work we focus on developing algorithms that are applicable on large datasets. This is the first topic of the thesis.

After computing the concept lattice, it is presented to the user for examination. The quality of the communication of the findings of the analysis is of great importance in the data analysis process. The proper visualization of the line diagrams allows to structure the data and to exhibit the dependencies that exist between different attributes in the data. Finding a good layout for the line diagram of the lattice can be a hard task. Layout algorithm are therefore the second topic of the thesis.

Research motives and contributions

The interest for the application of Formal Concept Analysis to datasets that include tens or hundreds of thousands of objects has been one of the main driving forces for the research that is described in this thesis.

We have developed efficient algorithms for the calculation of the set of all concepts that also can easily be adapted to other tasks, namely, to the task of calculating the Hasse diagram and of enumerating of all concept lattice elements.

For the first time, in this thesis the application of the Binary Decision Diagrams for the computation of the set of all intents is explored. It is shown, that algorithms using this representation perform extremely well for the case of dense contexts.

Also we have developed new algorithms for drawing line diagrams of con-

cept lattices. The best of them produce drawings of the lattices that are of high quality and aesthetically pleasant.

Outline

The thesis is organized as follows.

In Chapter 2 we introduce the basic notions of Formal Concept Analysis and describe the relationship of FCA and other knowledge representation formalisms.

In Chapter 3 we consider the problem of computing the set of all concepts and propose novel algorithms for the solution of this problem. We evaluate their characteristics theoretically and carry out an extensive experimental comparison with existing algorithms.

After that we consider how to use implicit representations, namely Binary Decision Diagrams, in order to calculate the set of all intents. Several novel algorithms are presented and their excellent performance for the case of dense contexts is demonstrated in comparison with other algorithms.

Knowledge of the characteristics possessed by the lattices produced by datasets that are typical for application areas are of importance for the development and optimization of algorithms. That is why in Chapter 4 we perform several experiments. First, we explore the distributions of the set of all concepts in dependence of the cardinality of the concept intent and extent. The results obtained in these experiments give rise to the question whether a preferable computation strategy exists for the JSM-method of automatic hypothesis generation, that is investigated next. And at last, the empirical exploration of the effects of the applications of lattice structure preserving transformations to real-world datasets is carried out.

In Chapter 5 the algorithms for visualizing concept lattices are presented. We present an algorithm for drawing concept lattices that minimizes the number of edge crossings and a force-directed algorithm for drawing concept lattices. The comparison shows that the first approach is a strong competitor with other algorithms and often produces aesthetically pleasant drawings.

Finally, in Chapter 6, we summarize the main contributions of the thesis and outline directions for future research.

Chapter 2

Formal Concept Analysis

2.1 A short introduction

A Formal Concept Analysis (FCA) is a branch of lattice theory that was proposed by Rudolf Wille in 1981 [82]. Sometimes it is also regarded as an applied section of lattice theory.

In this work we will be interested in developing efficient algorithms for performing basic tasks in FCA and using ideas from FCA in the field of data analysis and visualisation. In this chapter we will introduce the necessary terminology of FCA and discuss FCA from the knowledge representation (KR) point of view.

2.2 Basic terminology

One of the forms of data representation in FCA is called *formal context*. Formally it is defined as follows.

A triple (G, M, I) is called a formal context, if G and M are sets and $I \subseteq G \times M$ is a binary relation between G and M . The elements of G are usually called objects and the elements of M attributes.

Essentially, a (formal)¹ context is an object-attribute table, in which, if object o has attribute a , there is a cross in the cell on the intersection of the row corresponding to object o and the column corresponding to attribute a .

An example of a formal context is shown in Fig. 2.1.

When analysing such a table, it is quite natural to ask several questions:

1. If some attribute or a set of several attributes are given, what are the objects that have all these attributes (if any)?

¹also called unary

	Soft	Strong	Warm	Sparkling	With caffeine
Mulled wine		×	×		
Coke	×			×	×
Tee	×		×		×
Coffee	×		×		×
Mineral water	×				

Figure 2.1: Example of formal context “Drinks”

2. If some set of objects is given, what are the attributes common to all objects from the set?
3. If some conjunction of attributes is given, are there any other attributes that always occur in the table together with these attributes?
4. If some set of objects is given, are there any other objects that have all attributes that are common to the given set of objects?

In order to answer these questions so called *derivation operators* were defined in FCA:

- Let $A \subseteq G$. Then $A' = \{m \in M \mid gIm \forall g \in A\}$.
This operator sets up a correspondence between a set of objects A and the set of all attributes that occur in every object of A .
- Let $B \subseteq M$. Then $B' = \{g \in G \mid gIm \forall m \in B\}$.²
This operator establishes a correspondence between a set of attributes B and the set of all objects from G that possess all attributes of B .

Examples of results of derivation operators applications to some elements of the context “Drinks” are shown in Fig. 2.2. The application of the corresponding derivation operators allow immediately to answer question 1 and 2, and the application of the composition of two derivation operators (i.e. from G to M and then back to G , and from M to G and back to M) allows to answer questions 3 and 4. Operators, obtained as a result of the composition of operator $'$, i.e. operator $''$, are *closure operators*.³

²Technically, the first and second operator are different and can be denoted using different symbols. But in the mainstream FCA literature such notation is widely used and that is why we adopt it here.

³Formally, a closure operator is defined as follows: A *closure operator* ϕ on some set M is a map, assigning a *closure* $\phi X \subseteq M$ to each set $X \subseteq M$ that is:

Entity	Applied operator	Result
object Tea	'	$\text{Tea}' = \{\text{Soft, With caffeine, Warm}\}$
attribute Strong	'	$\text{Strong}' = \{\text{Mulled wine}\}$
object Tea	"	$\text{Tea}'' = \{\text{Tea, Coffee}\}$
attribute Strong	"	$\text{Strong}'' = \{\text{Strong, Warm}\}$

Figure 2.2: Results of derivation operators application to some elements of formal context “Drinks”

If one has some set of attributes $B \subseteq M$ then its closure will include all attributes that always occur in objects of the context that have all attributes from B .

A *formal concept* is a pair (A, B) where $A \subseteq G$, $B \subseteq M$ and $A' = B$, $B' = A$. A is called the *extent* and B is called the *intent* of the concept (A, B) .

It should be noted that a concept description is redundant, because, when the object set A is given, then the attribute set B can be easily computed from it and vice versa.

When a formal context is represented as a cross table, then every formal concept (A, B) corresponds to a maximally filled rectangular subtable of the context table, with row set A and column set B .

For example, some of the concepts of context “Drinks” are:

- $(\{\text{Mulled wine}\}, \{\text{Strong, Warm}\})$
- $(\{\text{Tee, Coffee}\}, \{\text{Soft, Warm, With caffeine}\})$
- $(\{\text{Tee, Coffee, Coke}\}, \{\text{Soft, With caffeine}\})$

Between concepts of a context a partial order can naturally be established by means of the following *subconcept-superconcept* relation

$$(A_1, B_1) \leq (A_2, B_2) \iff A_1 \subseteq A_2 (\iff B_2 \subseteq B_1).$$

The relation \leq is called a *hierarchical order*.

Example:

$$(\{\text{Tee, Coffee}\}, \{\text{Soft, Warm, With caffeine}\}) \leq (\{\text{Tee, Coffee, Coke}\}, \{\text{Soft, With caffeine}\})$$

monotone: $X \subseteq Y \rightarrow \phi X \subseteq \phi Y$

extensive: $X \subseteq \phi X$

idempotent: $\phi X = \phi \phi X$

The set of all concepts of the context (G, M, I) , ordered by the subconcept-superconcept relation \leq , is called a *concept lattice* of the context (G, M, I) and denoted as $\underline{\mathcal{B}}(G, M, I)$.

One part of the main theorem of the Formal Concept Analysis states that the concept lattice $\underline{\mathcal{B}}(G, M, I)$ is a complete lattice, in which the infimum and supremum are given by the following formulae:

$$\bigwedge_{t \in T} (A_t, B_t) = \left(\bigcap_{t \in T} A_t, \left(\bigcup_{t \in T} B_t \right)'' \right)$$

$$\bigvee_{t \in T} (A_t, B_t) = \left(\left(\bigcup_{t \in T} A_t \right)'', \bigcap_{t \in T} B_t \right).$$

The second part of theorem states that for every complete lattice the corresponding context can be build. The concept lattice of this context is isomorphic to the original lattice.

The *filter* of an element $a \in \underline{\mathcal{B}}(G, M, I)$ is the set $\{x \in \underline{\mathcal{B}}(G, M, I) \mid a \leq x\}$. Similarly, the *ideal* of an element $a \in \underline{\mathcal{B}}(G, M, I)$ is the set $\{x \in \underline{\mathcal{B}}(G, M, I) \mid x \leq a\}$.

To every object (resp. attribute) of a formal concept corresponds a so-called object (resp. attribute) concept. The object concept for an object g is a minimum (i.e. most specific) concept of the lattice that contains object g in its extent. The attribute concept for an attribute m is the maximal concept that contains attribute m in its intent. The correspondence between an object (resp. attribute) and its concept is set with the help of a special mapping γ (resp. μ). Let attribute m and object g are given. Then the mapping $\gamma : G \mapsto \underline{\mathcal{B}}(G, M, I)$ is defined as $\gamma g = (g'', g')$ and the mapping $\mu : M \mapsto \underline{\mathcal{B}}(G, M, I)$ as $\mu m = (m', m'')$.

2.3 Line diagrams of concept lattices

Usually for visualizing a lattice a technique called *Hasse diagram* is used. Instead of the visualization of the whole partial order relation \leq only its transitive reduction \triangleright is drawn. Usually the lattice unit (top, \top) element is drawn at the top of the drawing and lattice zero (bottom, \perp) element is drawn at the bottom of the drawing.

If every concept was marked on the Hasse diagram by the corresponding extent and intent, then this would lead to a great cluttering of the picture (see, for example, Fig. 2.3). In order to overcome this problem the special technique called *reduced labeling* is used.

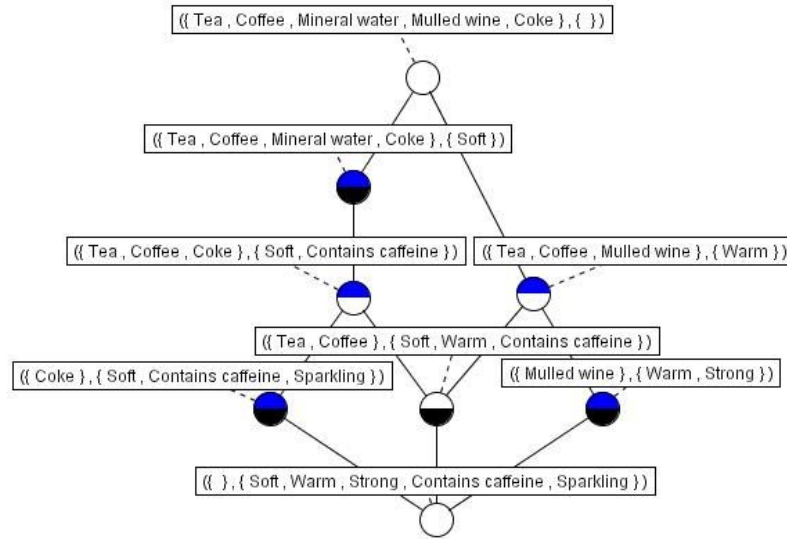


Figure 2.3: Line diagram of the concept lattice with full labeling for the context “Drinks”

In it attribute concepts are labeled with names of attributes and object concepts are labeled with names of objects. The reduced labeling does not lead to a loss of information. The intent and extent of any concept can be read off from the diagram as follows: the intent of a concept contains those attributes that can be reached from the concept node by ascending paths, going from this node to the unit concept node and the extent consists of all objects that can be reached by descending paths from the concept node to the zero concept node. For the diagram of the same lattice with reduced labeling see Fig. 2.4.

In the FCA literature for a lattice Hasse diagram very often the name *line diagram* is used.

The original context, from which the concept lattice was computed, can be easily restored from the line diagram. This can be done using the following formula: $gIm \Leftrightarrow \gamma g \leq \mu m$.

It should be noted that, although the context can be restored from a concept lattice, another, smaller context can exist, the lattice of which is isomorphic to the concept lattice of the original context. There are two ways of transforming a context that do not change the structure of a concept lattice. The first is the removal of equivalent attributes (i.e. attributes that occur ex-

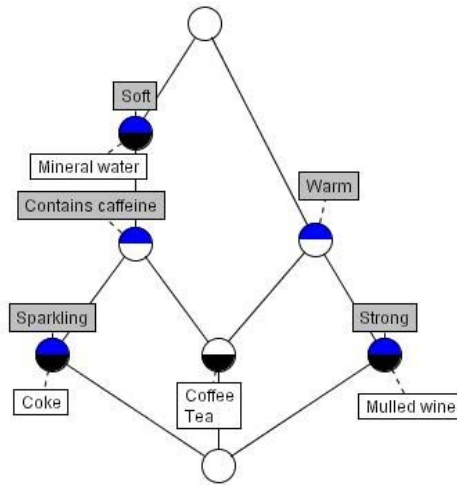


Figure 2.4: Line diagram of the concept lattice with reduced labeling for the context “Drinks”

actly in the same set of objects) and of equivalent objects (that possess the same set of attributes). The second one is the removal of attributes that can be expressed as a conjunction of other attributes, and of objects, the intents of which can be expressed as a combination of other objects. When one removes all such objects and attributes from a given finite lattice, then one gets a so-called *reduced context*, which is unique up to isomorphism. A reduced context is the smallest context, the concept lattice of which is isomorphic to the given concept lattice.

The procedure that finds a reduced context is called *context reduction*. It can be performed on contexts in polynomial time. Also, one can select irreducible objects and attributes from the Hasse diagram of a lattice. An element that has only one direct upper neighbour is called *join-irreducible* and an element that has only one direct lower neighbour is called *meet-irreducible*. The set of irreducible attributes can be obtained by selecting one attribute from the attached attributes for each join-irreducible element. Analogously, this can be done for irreducible objects, whereby only meet-irreducible elements have to be used.

2.4 Implication sets

In Section 2.2 we stated four typical questions that could be asked when analysing a context. Another typical question is: what are the dependencies between the attributes of the context?

One of the most well known for human and frequently used form of dependencies are so-called “if-then” dependencies or implications.

The implications that exist between attributes can be read off from the line diagram. The implication of the form $A \rightarrow m$ holds if and only if $\mu m \leq (A', A'')$.⁴

The number of implications that are valid in a context can be extremely big. For example, for a context with only one object that possesses all attributes the number of valid implications is equal to 2^{2^m} , where m is the number of attributes in the context. At the same time, all these implications follow from the single implication $\emptyset \rightarrow M$.⁵

That is why it is desirable to have some smaller set of implications that is:

- sound, i.e. it contains only valid implications,
- complete, i.e. it allows to infer all valid implications,
- not redundant, i.e. no one implication can be dropped without losing the property of completeness.

There are several implication sets that describe all dependencies that hold in a particular context. The most frequently used are the following ones:

1. the set of implications with a minimal premise;
2. the set of proper implications;
3. the Duquenne-Guigues set of implication.

A set of attributes $A \subseteq M$ is called the *minimal generator* of a closed set C , if $A'' = C$ and there is no strict subset $A^* \subset A$ such that $A^{*''} = C$.

The set of implications with a minimal premise contains implications of form $A \rightarrow A'' \setminus A$, where A is the set of minimal generators that are not equal to their closure.⁶

The main advantage of a set of implications with minimal premises is that it provides for each non-trivial conclusion the information about the minimal

⁴A general implication $A \rightarrow B$ holds iff $A \rightarrow m$ holds for all $m \in B$.

⁵This example is adapted from [33].

⁶This set of implications is also called *generic basis* in [4].

combinations of the attributes leading to this conclusion. Another advantage is that this set of implications can be easily transferred to the context with a modified set of attributes (i.e. some attribute was added or dropped).

The main drawback of such a set is its big redundancy.

The first attempt to overcome this deficiency was the *set of implications with proper premises* [34]. It is obtained from the set of implications with minimal premises by removing all implications that trivially follow from others or can be obtained with the help of the following inference rule: $A \rightarrow B, C \rightarrow D \models AC \rightarrow BD$. In the formulae AC is a shortcut for the set of attributes $A \cup C$ containing all attributes of A and C . This set of implications is also redundant.

The Duquenne-Guigues base of implications [22] is a sound, complete and irredundant base of implications. It contains the minimal possible number of implications. (This does not mean that it is minimal in terms of rules complexity.)

The Duquenne-Guigues base of implications is defined with the help of the notion of a *pseudoclosed set*. Let ϕ be a closure operator on the finite set M . Then a set $X \subseteq M$ is called *pseudoclosed* if and only if:

- $X \neq \phi X$;
- if $Y \subset X$ is a pseudoclosed proper subset of X , then $\phi Y \subset X$.

Let M be finite and $"$ be a closure operator on M . Then the set of implications

$$\{P \rightarrow P'' \setminus P \mid P \text{ is pseudoclosed}\}$$

is called the *Duquenne-Guigues base of implications*.

Till now we were talking about the strict implications. In such implications if some object of a context contains the attributes of the premise of implication $A \rightarrow B$ among its attributes, then it also necessarily contains the attributes of the conclusion of implication among its attributes.

Very often in practice also the other kind of dependencies is interesting, namely so-called *partial implications* [52], which hold almost for all objects. Partial implications are also known under the name “association rules” [1].

The association rule is the implication of the form $A \rightarrow B$, where $A \subseteq M, B \subseteq M$ and $A \cap B = \emptyset$. The association rule is usually characterised by the two parameters, namely *support* and *confidence*. The support of the association rule $A \rightarrow B$ is defined as

$$\text{support}(A \rightarrow B) = |(A \cup B)'|,$$

i.e. the number of objects that contain both the attributes of premise and conclusion of the association rule. The confidence of the association rule $A \rightarrow B$ is defined as

$$\text{confidence}(A \rightarrow B) = \frac{|(A \cup B)'|}{|A'|}.$$

The confidence defines which percentage of objects that contain premise of the association rule also contain the conclusion of the association rule.

2.5 Relationship between three kinds of knowledge representation in FCA

Here we will summarize the information about the basic forms of knowledge representation in the Formal Concept Analysis and describe relationships between them.

The basic types of knowledge representation in FCA are:

- object-attribute table, i.e. the formal context;
- concept lattice;
- implications and partial implications sets.

The relationships between them can be described as follows:

- The concept lattice is a univocal data-lossless transformation of a formal context. Each element of the concept lattice can be viewed as representing some similarity that exists among the members of some subset of objects. The elements of a concept lattice represent all similarities that exist between object representations in the context.
- There is an infinite number of contexts that produce an isomorphic concept lattice. Among them exists a unique (up to isomorphism) minimal one (in the number of objects and attributes) which is called reduced context.
- From a context the set of all implications that hold in it can be calculated. There are several sets of implications of practical interest.
- The set of all fixed points of a set of implications that hold in a context also forms a lattice isomorphic to the context concept lattice. The object-reduced context of such a lattice is a minimal context, the set of

implications of which is equal to the set of implications of the original context. The transformation from a context to the set of implications that hold in it leads to a loss of quantitative information about the frequencies of attribute occurrences in the context.

- The set of rules that preserves all information contained in a context consists of the base of exact implication rules and the base of partial (or inexact) ones that are known as association rules.

The graphical representation of the interrelationship is shown in Fig. 2.5.

2.6 FCA and knowledge discovery in databases

Our interest in FCA arises from the tight interrelationship that exists between several methods of machine learning, data mining and knowledge discovery in databases and Formal Concept Analysis.

This interrelationship can be described as follows:

1. Concept lattices are isomorphic to search spaces that are used in several tasks of data mining and machine learning, namely association rules [1] and generalized rules discovery [81], contrast set discovery [6, 80], search for interesting subgroups [43] and JSM method of hypothesis formation [25, 30].

FCA seems to be well suited for reasoning about methods that are based on complete search.

2. Using methods from FCA allows to analyse data that have attributes of several types, such as nominal, ordinal, or more complex ones.
3. The usage of lattice-based scales on values of attributes allows to consider not only the influence of one specific value of attribute, but also combinations of several values. Such a specification allows to elaborate in detail exactly what combinations are considered interesting and constrain the amount of the performed search.
4. On the basis of the main theorem of FCA any complete finite lattice (without another restrictions on the type of the lattice) corresponds to some formal context. That is why it seems possible to transfer results obtained for concept lattices to more complex formalisms in cases when the search space is finite.

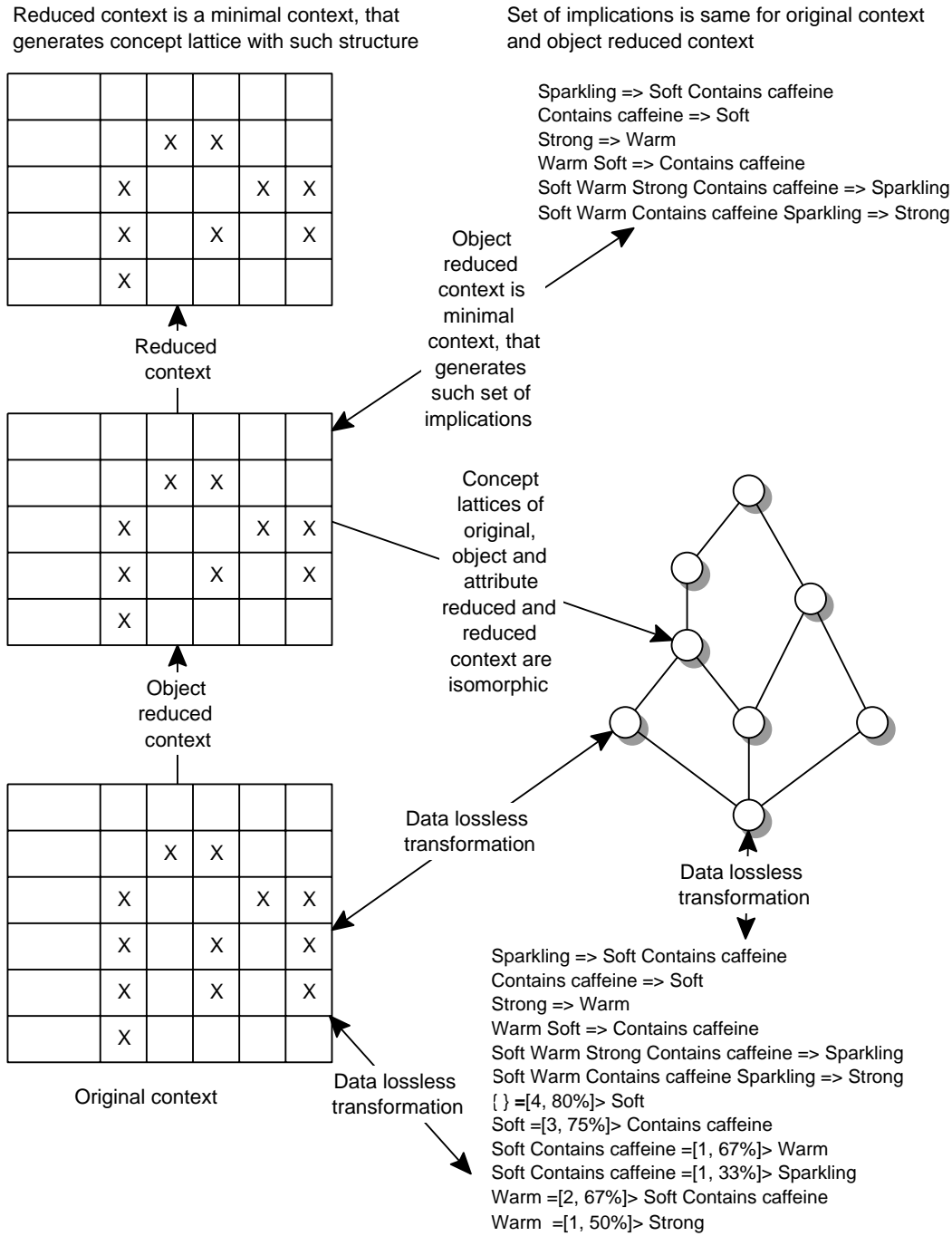


Figure 2.5: Relationship between different forms of knowledge representation in FCA. The $\{ \}$ is used for denoting the empty set. The notation $= [s, c\%] >$ is used for association rules, where s is the support of an association rule and $c\%$ is the confidence.

2.7 FCA from the viewpoint of KR

From the viewpoint of knowledge representation Formal Concept Analysis is one of the simplest formalisms for representing knowledge in terms of the expressive power. Contexts correspond to simple attribute-object tables. Implications have equal expressive power to propositional horn clauses.

There is a close connection between FCA and description logics (DL). Description logics are the formalisms that appeared from attempts to create a solid foundation for semantic networks [54]. Usually, they are used for the description of terminological systems. The basic building blocks of any description logic system are the set of primitive concepts and relations⁷ between concepts, and some set of constructors that are used in order to create new concepts from existing concepts and roles.

Features that distinguish description logics from some other knowledge representation formalisms are the availability of a clear Tarski-style semantics and a very deep understanding of the computational tradeoffs related with the cost of the usage of different constructions for creating new concepts.

Description logics are usually constrained in expressive power to decidable subfragments of first order logic.

The specification of a description logic knowledge-based system is divided into two parts: TBox and ABox. The TBox is used for the description of the terminology of a problem domain and the ABox is used for the specification of properties of individuals and objects occurring in the problem domain.

The relation between Formal Concept Analysis and Description Logic is the following: it is possible to set in correspondence to Formal Concept Analysis a very simple description logic without relations.

This description logic has the following formation rules:

- Axioms are of the form:

$$CN \sqsubseteq C \mid CN \doteq C$$

where CN is a concept name, C is a concept expression and '|' serves for listing alternatives.

- Concept expressions are of the form:

$$CN \mid C \sqcap D$$

where CN is a concept name, C and D are concept expressions.

⁷usually, but not necessary, binary

Let a context (G, M, I) be given. Then the mapping ϕ of the context to the description logic knowledge base $\mathcal{KB}(\text{TBox}, \text{ABox})$ is built as follows:

1. For each attribute $m \in M$ add the concept $\phi(m)$ to the TBox.
2. For each object $g \in G$ add the instance $\phi(g)$ to the ABox.
3. For each element $(g, m) \in I$ add the assertion $\phi(m)(\phi(g))$ to the ABox.
4. If there is some set $\text{Bl}(G, M, I)$ of background implications having the form $A \rightarrow B, A \subset M, B \subset M$ that are valid for all possible extensions of the context, then:
 - If there is an implication with the empty premise⁸ $\emptyset \rightarrow B$ in $\text{Bl}(G, M, I)$, then add to TBox the following axiom

$$\top \doteq \phi(m_1) \sqcap \dots \sqcap \phi(m_k),$$

where $m_1 \in B, \dots, m_k \in B$.

- For each premise of an implication $A \rightarrow B$ of $\text{Bl}(G, M, I)$ having cardinality greater than one, introduce the corresponding terminological axiom

$$\phi(A) \doteq \phi(m_1) \sqcap \dots \sqcap \phi(m_k), m_1 \in A, \dots, m_k \in A,$$

where $\phi(A)$ denotes a unique composite name for the description of the DL-concept in which the conjunction of attributes A maps. If A consists of one attribute m , then $\phi(A) = \phi(m)$.

- For each implication $A \rightarrow B, A \neq \emptyset$, add to the TBox the following terminological axiom

$$\phi(A) \sqsubseteq \phi(m_1) \sqcap \dots \sqcap \phi(m_k), m_1 \in B, \dots, m_k \in B.$$

Such a description logic has a smaller expressive power than the simplest structural description logic \mathcal{FL}^- .

But one should point out that the tasks solved by FCA on the one side and by description logics on the other are different and complimentary ones. Description Logics aim to describe a general ontology of some domain that will hold in all possible interpretations. One can say that description logics

⁸It is assumed that there can be at most one such an implication, otherwise all implications having the form $\emptyset \rightarrow m_1, \dots, \emptyset \rightarrow m_k$ can be replaced by one, namely $\emptyset \rightarrow m_1 \wedge \dots \wedge m_k$.

first start from the definition of primitive concepts and then try to specify the description of the domain that would be full and adequate.

Formal Concept Analysis, on the other side, starts from available data and tries to find the dependencies that hold in a particular dataset (context). The dependencies completely describe the available data, but they need not necessarily be true in general.⁹

For the extension of Formal Concept Analysis, which maps to a more expressive description logic \mathcal{ALC} , see [63].

2.7.1 Mapping between terminology of FCA and DL

Unfortunately, as it often happens in science, due to independent roots of development of FCA and DL, there is a terminological clash between them. The same notions are described by using different names and notions with similar names have different meanings.

The notion of context attributes in FCA corresponds to the notion of atomic concepts in DL. They can be understood as a unary predicate. The conjunction of concepts in DL corresponds to the infimum of attribute concepts in concept lattices in FCA.

The intent of a formal concept in FCA can be set in correspondence to the concept expression in DL and the extent of a formal concept can be set in correspondence to the extension of this concept expression in one of the possible models of the DL based knowledge-based system.

In DL the notion of an attribute is sometimes used for functional roles.

Description Logics have a strict distinction between the TBox containing purely intensional definitions of concepts and roles, and the ABox that provides information about individuals. In FCA this distinction is blurred, because usually analysis starts from available data, and one of the tasks of the analysis can be to find a set of dependencies that exist between attributes. This is the reason why there is no explicit TBox in FCA, although in some usage scenarios it is possible to view the set of background implications as a TBox.

⁹If one wants to find a set of general implications, then one should use the procedure of attribute exploration in FCA, provided that an expert is available who can give valid answers to all questions concerning the domain, such as whether some particular implication always holds in domain and provide a counterexample otherwise.

Chapter 3

FCA Algorithms

One of the important components of any tool that uses Formal Concept Analysis is the algorithms. Their efficiency plays a crucial role in applications. Previously most of the algorithms for Formal Concept Analysis were compared and tested on small datasets. That is why this chapter is devoted to the development of more efficient algorithms for Formal Concept Analysis, that are applicable on large datasets. In Section 3.1, we describe the different scenarios of using algorithms in concept lattices. Then we consider the desirable properties of such algorithms in Section 3.2. After that we consider a simple approach for the calculation of the set of all concepts and improvements to it (Section 3.3), that eventually lead to algorithm called Grail (Section 3.4). In Section 3.5 the computational complexity of the developed algorithms is estimated. In Section 3.6, we perform an experimental comparison of the developed algorithms with other algorithms for the calculation of the set of all concepts. Next, the question of using implicit representations in algorithms is explored in Section 3.8. The chapter is closed with the review of the related work.

3.1 Usage scenarios

There are several different usage scenarios, in which algorithms for the exploration of a concept lattice¹ (resp. search spaces) are useful. Among them one can separate the following ones:

1. Systems that are aimed for researchers in lattice theory. On the one

¹Under the algorithms for exploration of concept lattice we understand algorithms that allow to generate the whole concept lattice (resp. line diagram) or some part of it and to derive related concept lattices (i.e. differing in some objects (resp. attributes)) from already existing one.

side here one needs the most extensive set of algorithms for work with lattices. From the other side, one can expect the dimension of contexts and lattices to be of modest size.

2. Document retrieval and classification systems [13, 48]. Here the number of documents and classification attributes can be quite big, but usually the whole concept lattice is not required. In such a system the user usually formulates and then refines queries based on the feedback from the system. The most important operation is the calculation of upper and lower neighbours for the element of the lattice on which the user is currently focusing.
3. Program analysis component inside a compiler or software engineering system (see, for example [67, 69, 49]). In such a task, the calculation of the whole concept lattice can be required, and the size of contexts can be up to several thousands of objects.
4. Data visualisation systems for visualising the content of databases [78]. In such a system line diagrams are set in correspondence to columns of database tables and the user can explore the distribution of objects, create combined views, and perform drill down and roll up operations. Lattices are usually of moderate size, and the main emphasis is put on optimising the queries and the amount of the performed queries to the database.
5. Data analysis (or mining) systems that allow the user to explore some dataset, perform search for different kinds of dependencies and then explore and refine the results in interesting sections. Here the requirements for the algorithms can be quite different:
 - One possible approach is to perform first a time-consuming operation of mining some part of the lattice (so called iceberg-lattice [72]) and then to allow the user to formulate, try and refine different queries on the lattice (arguments in support of this approach can be found in [38]).
 - Another approach is to store during the search only the most interesting points and then expand them and explore their neighbourhood on user demand.

In both these approaches there is a need for a procedure that allows to explore the lattice search space

There are several tasks that arise, when analysing data presented as a binary table. They are:

- Given a formal context (G, M, I) , calculate all elements of $\underline{\mathcal{B}}(G, M, I)$ (task of the construction of the set of all concepts).
- Given a formal context, calculate a Hasse diagram of $\underline{\mathcal{B}}(G, M, I)$.
- Given a user specified value of the minimal concept extent cardinality, find all concepts that satisfy this constraint (task of the generation of frequent closed itemsets used in association rule mining).
- Given some object, find other objects that are closest and determine attributes, in which they differ. (Such a task appears, for example, in online stores, such as amazon.com, when creating a recommendation for the user.)
- Given predefined hierarchies on attributes, find rules that are of maximal interest to the user (generalized association rule mining task [70]).
- Find a specified number of concepts that have a maximal value under a user-specified evaluation function (interesting subgroups discovery and generalized rule discovery task).

In the remainder of this chapter we will present algorithms for the solution of some of the aforementioned tasks.

3.2 Desirable properties of algorithms

In this section we will mention properties that are desirable for the search algorithms:

- Generation of any point in the search space only once.
- Maximal exploitation of already discovered information.
- Ease of incorporation of the search constraints into the algorithm.
- Ease of incorporation of background knowledge into the search algorithm.
- Ability to perform anytime search in order to provide quick feedback for the user.

```

procedure CalculateConceptSet
input   $(G, M, I)$  // context for which concept set is calculated

 $(G, G') := \text{CalculateUnitElementOfLattice}$ 
 $\text{Concepts} := \{(G, G')\}$ 
CalculateChildren( $(G, G')$ )

procedure CalculateChildren
input   $(A', A'')$  // parent concept

 $\text{ConceptsCandidates} := \text{FindSetOfChildrenCandidates}((A', A''))$ 
for each  $\text{Candidate} \in \text{ConceptsCandidates}$ 
  if  $\neg \text{AlreadyComputed}(\text{Candidate})$ 
     $\text{Concepts} := \text{Concepts} \cup \text{Candidate}$ 
    CalculateChildren( $\text{Candidate}$ )

```

Figure 3.1: The scheme of algorithm for the construction of the concept set

3.3 Construction of the set of concepts

First we will start with a simple approach to the construction of all elements of the concept set and then will analyse, how it can be enhanced, and check the usefulness of these extensions. After that we will consider, how the algorithm can be adopted for solving other related tasks.

One of the simple approaches to building a concept lattice is to generate first the unit element of the lattice and then proceed with the generation of its children, using some kind of traversal strategy.

The main concern when using such an approach is to generate each element of the lattice only once. This is achieved by storing in some way information about already generated elements. The scheme of such an algorithm is presented in Fig. 3.1.

It works as follows: in the procedure **CalculateConceptSet** the unit element of a lattice is calculated. This is always possible, because every concept lattice has at least one element. Then the unit element is added to the set of all concepts *Concepts* and the procedure **CalculateChildren** is called in order to generate all other concepts.

In the first step of the procedure **CalculateChildren** the set of all new candidates of concepts is computed by the procedure **FindSetOfChildrenCan-**

didates. The candidates set represents some subset of the set of concepts obtained from the concept (A', A'') by applying to it some neighbourhood operator (for example, adding one attribute to the concept intent and computing the intent closure). Then, if the candidate is generated for the first time, it is added to the set of concepts and the procedure `CalculateChildren` is called recursively with the new concept.

Such a scheme can be instantiated in different ways, namely by choosing a traversal strategy for the search space when generating a lattice and by choosing and/or adapting to the traversal strategy a memory mechanism for storing information about already generated concepts.

We will now present one mechanism that can be used when generating the lattice in a depth-first order. It is based on the observation that when some element of the concept lattice was generated by using an instantiation of the general scheme of this strategy, then the procedure `CalculateChildren` will not return before all elements of the ideal of this element are computed. The graphical demonstration of this fact is shown in Fig. 3.2.

In the picture the line diagram of the lattice of our favourite example from Fig. 2.1 is presented. The attribute names are chosen as the letters a through e . To each element of the lattice the label with the intent of the element is attached. All elements of the c attribute concept ideal are contained in the area, marked by the bold line. As one can see from the figure, the attribute c is contained in the intents of all its ideal elements.

This observation is equivalent to the fact that the intent of concept grows monotonously from the unit element of the lattice till the zero element of lattice. It allows to perform the check, whether some concept was already generated, locally. Although this does not change the asymptotic complexity of the procedure of the check of earlier concept generation, it allows to greatly reduce memory requirements for performing such a check. Namely, if one stores all already generated concepts in a trie² data structure, then the check for the earlier concept generation can be performed in time $O(m)$, where m is the number of attributes of the context. The drawback is the storage of all already generated concepts. When one is using the lattice ideal property, then the generation history can be stored by using a one bit-vector on each level of the implicit search tree. In this bit-vector the attributes used for the generation of a child concept from the current one are stored. Because for

²A trie (from **retrieval**) is a multi-way tree data structure for storing strings or ordered sets. There is one node in a trie for every common prefix of strings. General sets can be stored in a trie after defining some linear order on their elements. The complexity of checking, whether some string is contained in a trie is $O(m)$ or $O(m \cdot \log(m))$ (depending on the implementation), where m is the length of the string. A trie is also called a prefix tree.

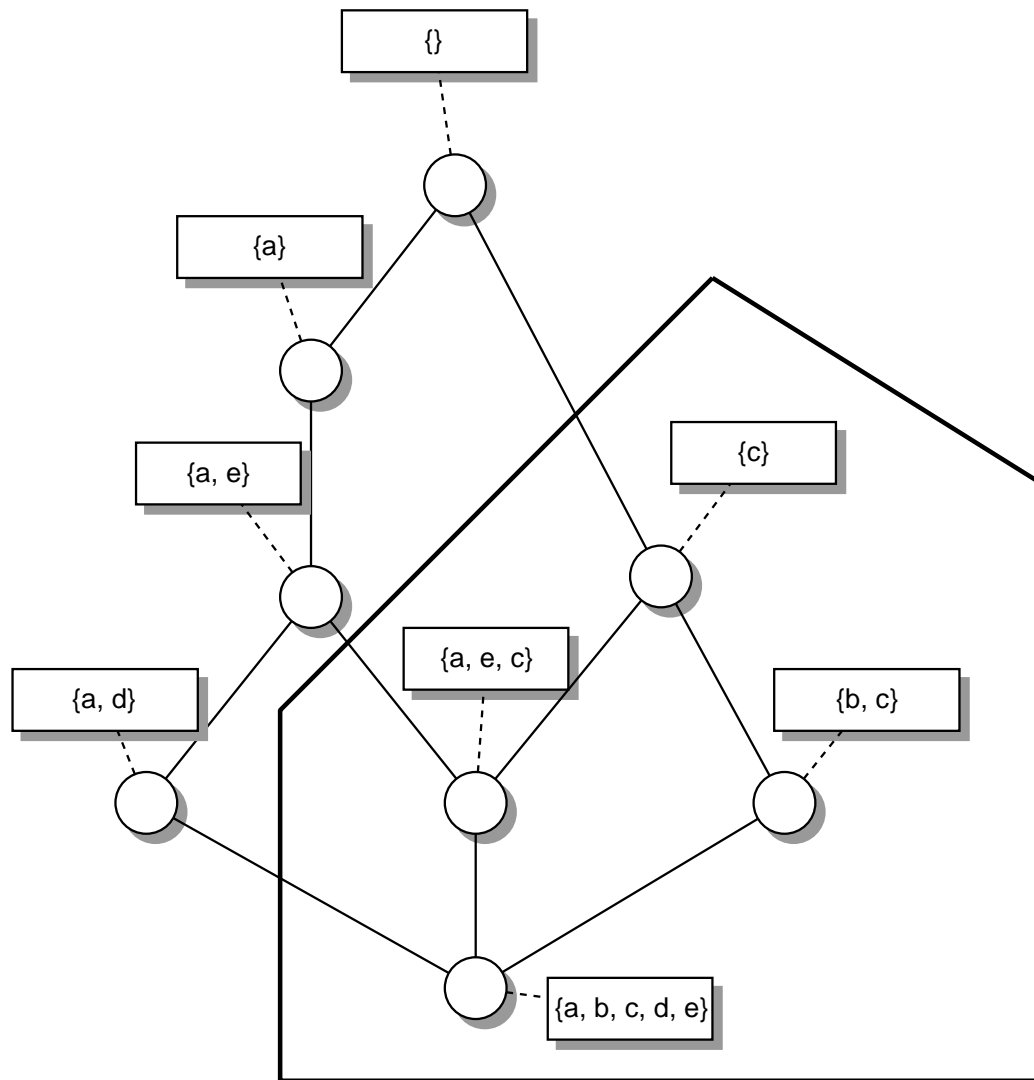


Figure 3.2: Demonstration of the lattice ideal property that allows to perform a local check of already generated concepts

such an attribute all the children were already computed the attribute has not to be contained in the intent of another new concept, generated from the current parent concept. The check is performed by testing, whether one of the elements of the difference between the child concept intent and the parent concept intent is already contained in the bit-vector.

The algorithm in which these ideas are implemented is presented in Fig. 3.3. It works as follows. At start, the unit element of concept lattice is computed (first three lines of the body of procedure `CalculateConceptSet`). Then, the procedure `CalculateChildren` is called in order to calculate all concepts that lie below the top element.

Variable *AlreadyComputed* serves the following two reasons:

1. it stores the history of the generation, and
2. it stores in an implicit way the list of attributes used to obtain new concepts from the existing one.

That is why an attribute can be contained in this variable due to the following reasons:

- it is already contained in the intent of the parent concept (i.e. the addition of such an attribute to the parent concept intent will not change it),
- it was already used in order to generate a child concept from a parent one and all children of the child concept were already computed.

Variable *Active* is the complement of *AlreadyComputed* and describes the set of all attributes that can be used in order to obtain a more specific concept than the current concept.

After the computation of *Active* from *AlreadyComputed*, the cycle (for each $a \in \text{Active}$) is performed in which all possible child concepts of the current one are calculated. They are obtained by adding to its intent one attribute and then computing the closure.

In the calculation the following variables are used:

- *NewExtent* (at end of the calculation it will contain the extent of the new concept),
- *NewIntent* (at end of the calculation it will contain the intent of the new concept).

We will demonstrate the execution of the algorithm on the example of the context from Fig. 2.1. In order to shorten the description, object names are

```

procedure CalculateConceptSet
input   $(G, M, I)$  // context for which concept set is calculated

 $A := M$ 
for each  $g \in G$ 
     $A := A \cap g^I$  // Calculating  $G'$ 
 $Concepts := (G, A)$ 
CalculateChildren(  $(G, A), A$  )

procedure CalculateChildren
input   $(A', A'')$  // parent concept
         $AlreadyComputed$ 

 $Active := M \setminus AlreadyComputed$ 
for each  $a \in Active$ 
     $NewExtent := \emptyset$ 
     $NewIntent := M$ 
    for each  $g \in A'$ 
        if  $a \in g^I$ 
             $NewExtent := NewExtent \cup \{g\}$ 
             $NewIntent := NewIntent \cap g^I$ 
     $Face = NewIntent \setminus A''$ 
    if  $Face \cap AlreadyComputed = \emptyset$ 
         $Concepts := Concepts \cup \{(NewExtent, NewIntent)\}$ 
        CalculateChildren (  $(NewExtent, NewIntent), AlreadyComputed \cup Face$  )
         $AlreadyComputed := AlreadyComputed \cup \{a\}$ 

```

Figure 3.3: Simple depth-first approach for the construction of the set of concepts

replaced by the numbers 1 through 5 and attribute names are replaced by the letters a through e . The picture that describes the execution of the algorithm is shown in Fig. 3.4. It is assumed that the selection of the next attribute is always performed in alphabetical order from a to e . The rectangular nodes present the elements of the concept lattice that were computed for the first time during the execution of algorithm. Inside such a node the concept and the values of variable *Active* and variable *AlreadyComputed* at the time of the call of **CalculateChildren** are presented. The variable *Active* represents the set of attributes that can be used to calculate child concepts of the current concept. The circles indicate the calculation of nodes which later turned out to be redundant, because the node was already computed.

The execution of the algorithm starts with the computation of the lattice unit element. In the example context there are no attributes that are common to all objects and the intent of the unit element is empty. The extent of the unit element is always equal to the set of all objects. After that the first call to the procedure **CalculateChildren** is performed. As the “parent concept” argument of the procedure the unit element of the lattice is passed on and the set *AlreadyComputed* = \emptyset . The set of active attributes *Active* is equal to the set of all attributes $\{a, b, c, d, e\}$.

The attribute a is selected as the first attribute to be used to obtain the more specific element from the unit element. The resulting concept is $(\{2, 3, 4, 5\}, \{a\})$. Then the procedure **CalculateChildren** is called in order to calculate the set of all children concepts of the new concept. The value of the variable *AlreadyComputed* in this call remains \emptyset .

In this call of **CalculateChildren** the attribute b is selected as the first extension attribute. The concept, obtained as a result of adding b to the intent of concept $(\{2, 3, 4, 5\}, \{a\})$ is the bottom one, because the attributes a and b do not occur together. As the intent of the bottom concept contains all attributes, the recursion terminates here. After that the attribute b is added to the *AlreadyComputed* set in the frame of procedure **CalculateChildren** for concept $(\{2, 3, 4, 5\}, \{a\})$. Then the calculation proceeds with the other attributes from the set *Active*. The eight circles in the picture show that this algorithm attempts to recalculate 8 elements out of 8. That is why the natural question arises: how one can avoid this waste of computational efforts?

The first idea is to try to exclude attributes that lie in the difference of the intents between the children concept and the current concept (represented in the variable *Face*). This is not always possible and can lead to a loss of completeness of the algorithm (i.e. some concepts can be missed). The reason for this phenomenon is that some other child of the current concept (A', A'') , for example $((A \cup \{b\})', (A \cup \{b\})'')$ can be more general than the

	a	b	c	d	e
1		×	×		
2	×			×	×
3	×		×		×
4	×		×		×
5	×				

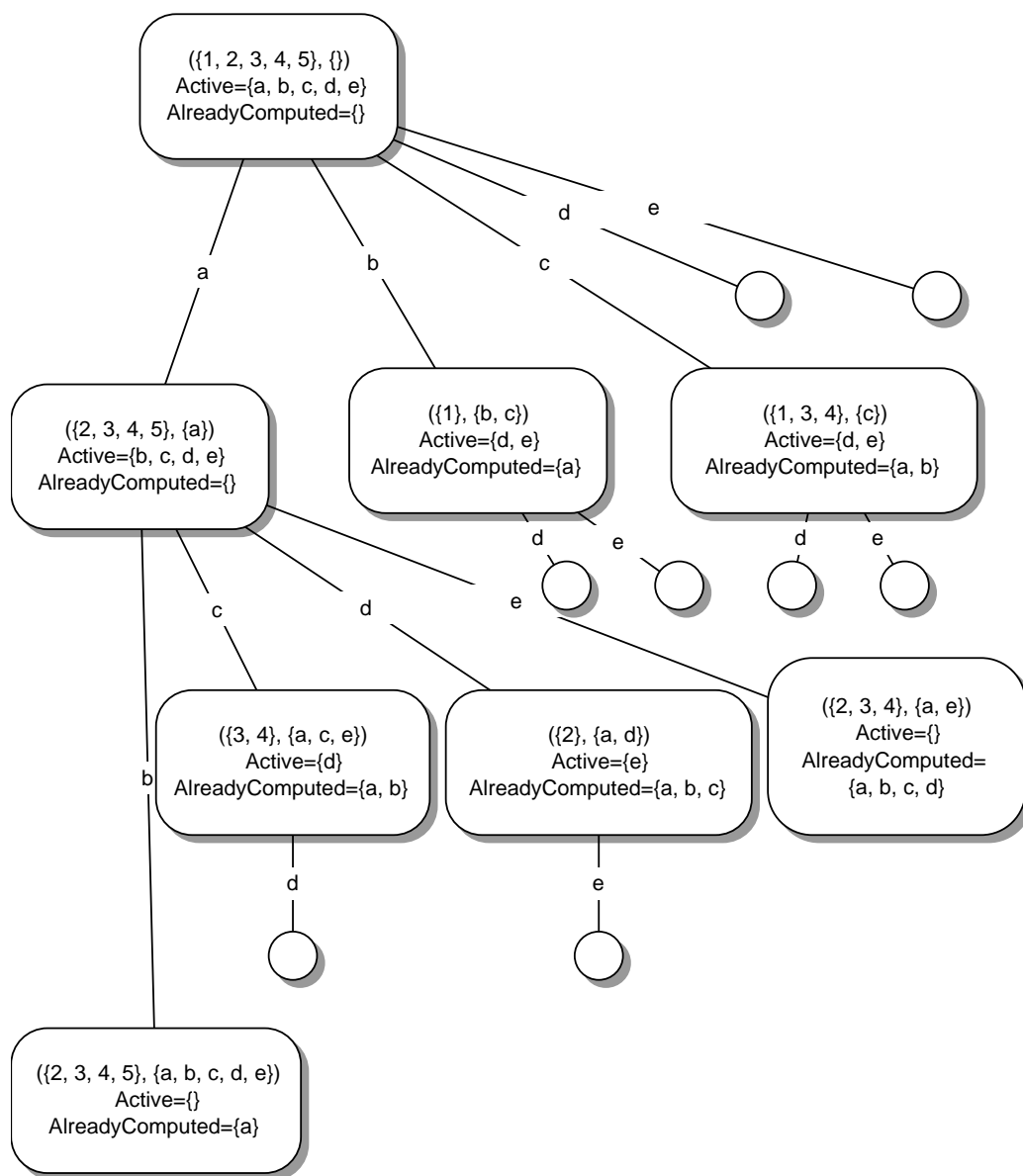


Figure 3.4: Example context and trace of the execution of the algorithm from Fig. 3.3 on it

child concept $((A \cup \{a\})', (A \cup \{a\})'')$. The next section will be devoted to a more detailed analysis of ways of improving algorithm.

3.4 Decomposition of context w.r.t. an attribute

In this section we will consider how the context can be decomposed with regard to a single attribute, and what the implications of this decomposition are for the calculation of the set of all concepts of context.

Let some attribute $a \in M$ of context be selected. The set of attributes of the context with respect to the current attribute can be decomposed into attributes that

- are possessed by all objects in the context, and the objects in this way do not depend on attribute a . This set is denoted as *Const*. Such attributes subsume any other attribute $b \in M \setminus \text{Const}$.
- subsume the selected attribute but are not constant. These attributes occur in all objects that occur with the current attribute, and there is at least one more object outside the extent of the attribute concept of attribute a , in which a subsuming attribute occurs, and one object, in which the attribute does not occur. We will denote this set as *Bigger*(a).
- are equivalent to the current attribute (i.e. these attributes occur only in the objects in which the selected attribute a occurs). We will denote this set as *Equivalent*(a).
- are strictly subsumed by the selected attribute (i.e. they occur only in objects, in which the selected attribute a occurs, and there is at least one object in which the selected attribute a occurs, and the subsumed one does not). This set will be denoted as *Children*(a).
- are incomparable with the selected attribute a , but occur together with the selected attribute in some object(s) of the context. We will denote this set of attributes as *OccuringIncomparable*(a).
- do not occur in any object from the context. This set of attributes will be denoted as *Empty* (which does not depend on the selected attribute a).

- are disjoint with the selected attribute and occur only in objects which lie outside the current concept. These attributes are also incomparable with selected attribute, but the property of disjointness has important computational consequences, for which reason we separate it here. Namely, if it is known that some attribute b is disjoint with the selected attribute a , then all objects of a 's extension can be ignored when computing the extension of b . We will denote this set of attributes as $Disjoint(a)$.

The most general case of such a decomposition is demonstrated in Fig. 3.5. The meaning of the symbols, used in the figure is as follows:

- X : all cells in this area have a cross;
- O : all cells in this area are empty;
- X, O : in this area, there is at least one filled cell for every attribute. For subsuming, subsumed or incomparable attributes it is required that there is at least one empty cell in this area. For disjoint attributes there is no such a requirement, i.e. it can be the case that all cells have crosses in this area.

Let some concept (A', A'') be given and for the selected attribute $a \notin A''$ hold. Then the decomposition can also be applied to the subcontext that is formed by objects from A' and attributes from $M \setminus (A'' \cup Empty)$.

We will denote the sets forming a decomposition as $Bigger(A'', a)$, $Equivalent(A'', a)$, $Children(A'', a)$, $OccuringIncomparable(A'', a)$ and $Disjoint(A'', a)$. Attributes of $Const$ will belong to the intent of every concept of the lattice and that is why there is no $Const$ member in the decomposition.

There are several implications of the decomposition of the subcontext. The simplest of them is the following one:

- All concepts that can be formed by adding some attribute b to the concept (A', A'') , where $b \in Children(A'', a)$, will also contain the attribute a . That is why it can be removed from the list of active attributes for the calculation of other children of the concept (A', A'') after all children concepts are calculated that lie in the ideal of the concept $((A'' \cup \{a\})', (A'' \cup \{a\})'')$.

A reformulation of this statement is the following:

- After all elements from the ideal of the concept $((A'' \cup \{a\})', (A'' \cup \{a\})'')$ are calculated, all other children of the concept (A', A'') not in the ideal

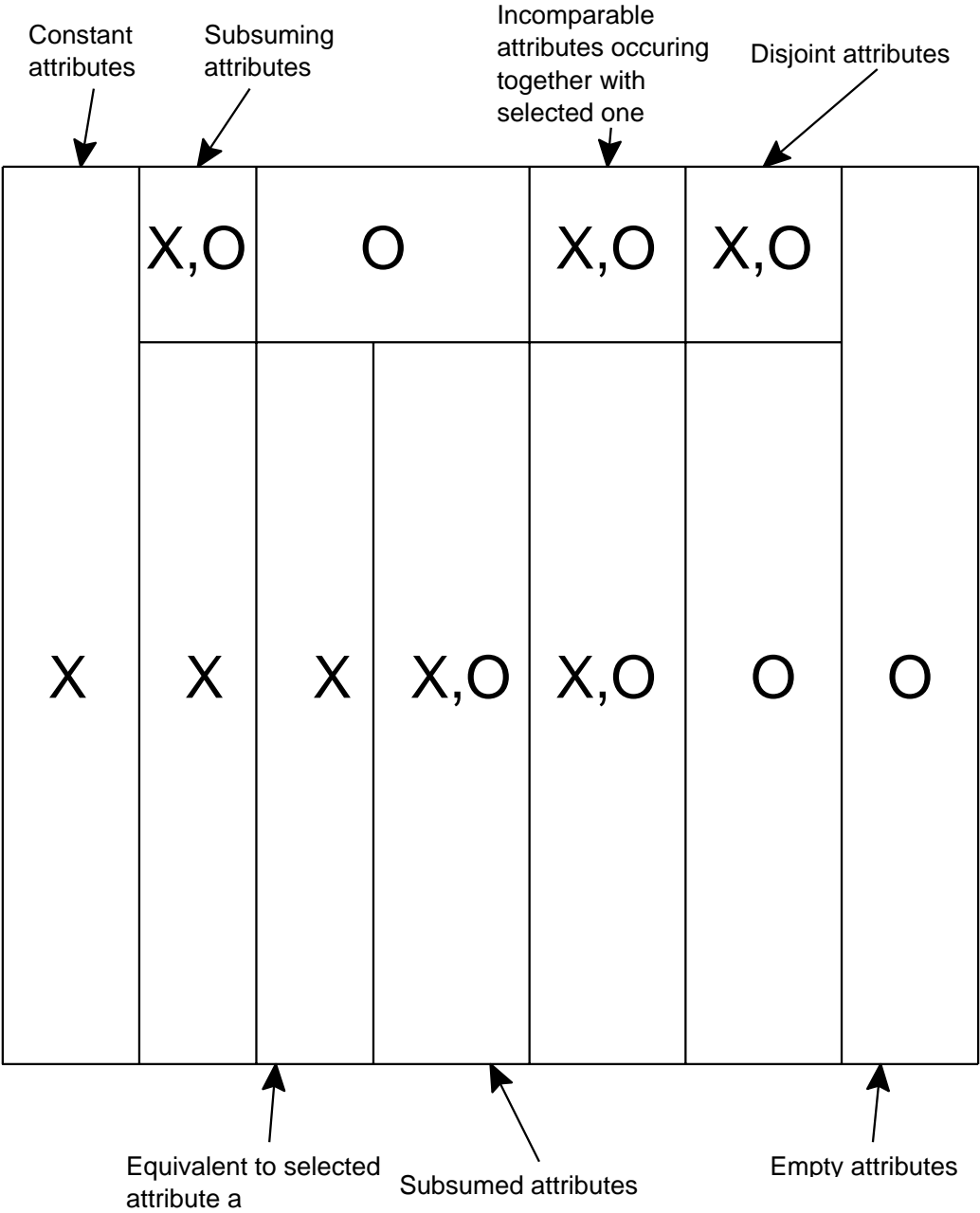


Figure 3.5: The decomposition of the context with respect to a selected attribute a

of $((A'' \cup \{a\})', (A'' \cup \{a\})'')$ can be obtained by considering the combination of only those attributes which are possessed by objects that lie in the difference between the extents of the concept (A', A'') and of the child concept $((A'' \cup \{a\})', (A'' \cup \{a\})'')$, i.e. in the set

$$ExtDiff(A'', a) = \{g \in G \mid g \in A' \setminus (A'' \cup \{a\})'\}.$$

This simple statement allows to propose the first optimization to the aforementioned algorithm. The pseudocode of the algorithm that implements this optimization is presented in Fig. 3.6.

The working of the algorithm is similar to the previous one. It differs in the following ways:

- During the calculation of the closure of the new children concept $((A'' \cup \{a\})', (A'' \cup \{a\})'')$ the additional set *Outer* is calculated. At the end of the calculation the set *Outer* contains all attributes that occur in the objects of set $ExtDiff(A'', a)$, i.e.

$$Outer(A'', a) = \{m \mid m \in g^I, g \in ExtDiff(A'', a)\}$$

- The set *Outer* is used in order to shorten the list of attributes that will be used when trying to calculate children concepts of current concept (A', A'') . Correctness of this optimization follows from the aforementioned statement. *Outer* is also used to expand the set *AlreadyComputed* by attributes that are not contained in *Outer*. This optimization shortens the list of attributes that can be used for the expansion of the next children of the current concept. This optimization is correct, as all concepts that can be formed from the current one by attributes outside *Outer* were already computed in the call of procedure **CalculateChildren**.

In case one is interested in the calculation of the whole concept lattice, the check for an earlier generation of the concept cannot be dropped because in this case the bottom element of the lattice, the intent of which is equal to the set of all attributes will be generated several times. In many tasks the bottom element of the lattice is not required in which case the algorithm can be changed in order not to generate the zero lattice element. In this case the check can be dropped altogether. For doing this, the detection of attributes that are disjoint with the selected attribute in the current subcontext ($Disjoint(A'', a)$) should be built into the algorithm and then all such attributes be excluded from set *Active* for the child of the current concept formed by addition of the selected attribute.

```

procedure CalculateConceptSet
input  ( $G, M, I$ ) // context for which set of concepts is calculated

 $A := M$ 
for each  $g \in G$ 
     $A := A \cap g^I$  // Calculating  $G'$ 
 $Concepts := (G, A)$ 
CalculateChildren( $(G, A), A$ )

procedure CalculateChildren
input  ( $A', A''$ ) // parent concept
         $AlreadyComputed$ 

 $Active := M \setminus AlreadyComputed$ 
for each  $a \in Active$ 
     $NewExtent := \emptyset$ 
     $NewIntent := M$ 
     $Outer := \emptyset$ 
    for each  $g \in A'$ 
        if  $a \in g^I$ 
             $NewExtent := NewExtent \cup \{g\}$ 
             $NewIntent := NewIntent \cap g^I$ 
        else
             $Outer := Outer \cup g^I$ 
     $Active := Active \cap Outer$ 
     $Face := NewIntent \setminus A''$ 
    if  $Face \cap AlreadyComputed = \emptyset$ 
         $C := C \cup \{(NewExtent, NewIntent)\}$ 
        CalculateChildren ( ( $NewExtent, NewIntent$ ),  $AlreadyComputed \cup Face$ )
         $AlreadyComputed := AlreadyComputed \cup (M \setminus Outer)$ 

```

Figure 3.6: Algorithm for the generation of the concept lattice with first optimization

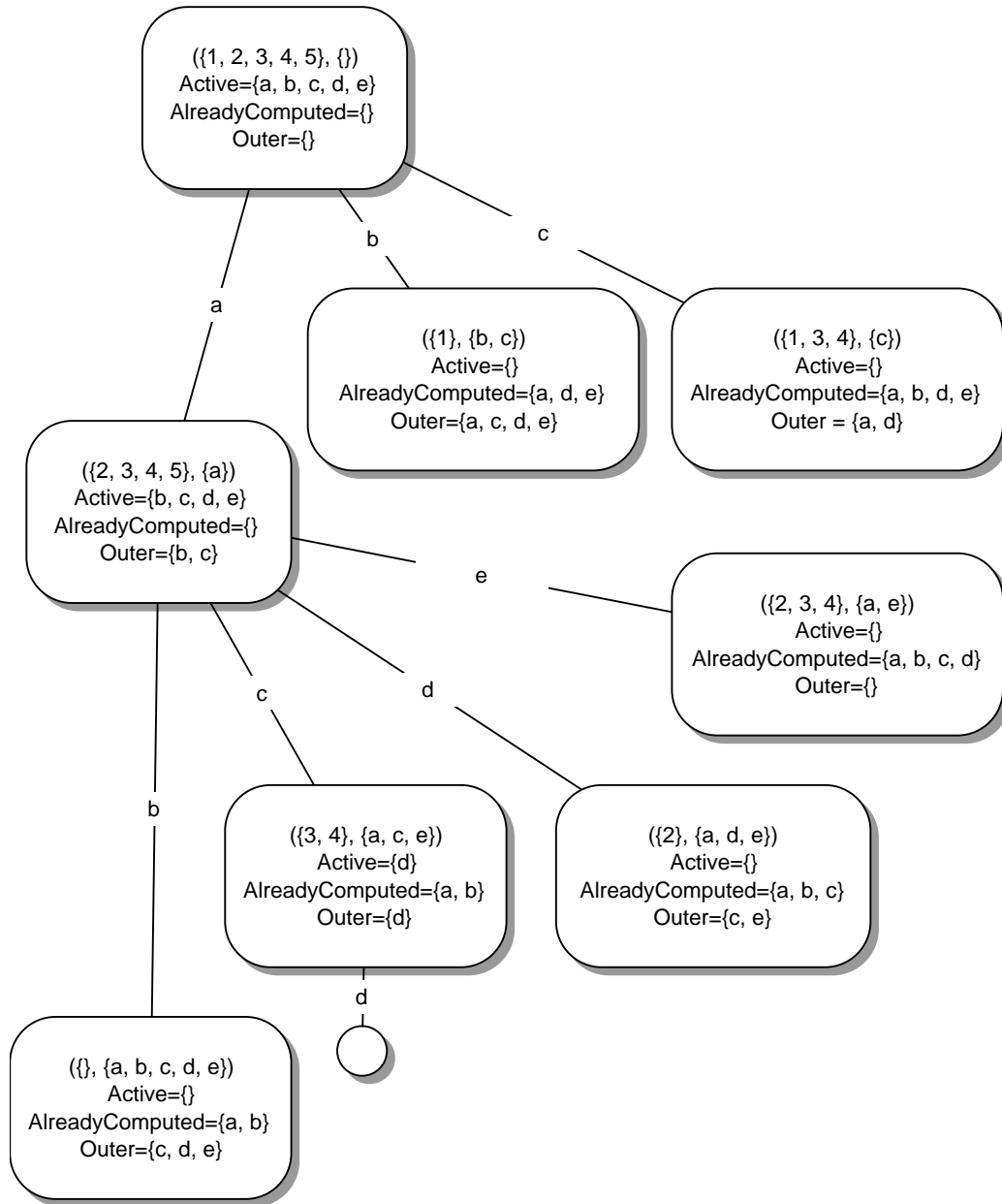


Figure 3.7: Trace of the execution of the algorithm in Fig. 3.6 for the example from Fig. 3.4

The trace of the execution of the algorithm for the example from Fig. 3.4 is shown in Fig. 3.7. As one can see by comparison with Fig. 3.4, the proposed optimization reduces the number of redundant calculations of already computed elements of the concept lattice in this example from eight to one, at the cost of the additional computation of the extra set *Outer*. This cost is usually smaller than the cost of redundant concept generation, although there are some cases, when this optimization slows down the algorithm, for example, for the case of the boolean context (G, G, \neq) .

3.5 Computational complexity of algorithms

In this section we consider the computational and memory complexity of the algorithms from Fig. 3.3 and Fig. 3.6.

3.5.1 General intractability of the task of computing all concepts

The number of concepts for the context (G, M, I) can be exponential in the size of the object or attribute sets. This can easily be seen with the example of the context of type (G, G, \neq) , for which the number of formal concepts is maximal and is equal to $2^{|G|}$.

Usually in the theory of the computational complexity *decision problems* (i.e. problems for which the algorithm should provide the answer “yes” or “no”) and *search problems* (i.e. problems for which the algorithm should provide some solution to the instance of the problem) are studied. The *enumeration problem* for a search problem is to determine for an instance of the problem how many solutions exist for it.

The enumeration problem of determining the number of elements in the concept lattice of a context is $\#\mathcal{P}$ -complete³ [46].

That is why there is little hope that a polynomial time algorithm for solving the task exists. In order to solve such a task, one can develop algorithms that are as efficient as possible and have a good performance in practice in the worst case or for relevant tasks.

³An enumeration problem belongs to the class $\#\mathcal{P}$ if there is a non-deterministic, polynomial-time Turing machine that, for each instance I of the problem, has a number of accepting computations that is exactly equal to the number of distinct solutions for instance I . A problem is $\#\mathcal{P}$ -complete (pronounced as Sharp-P-complete or Number-P-complete) if and only if it is in $\#\mathcal{P}$, and every problem in $\#\mathcal{P}$ can be reduced to it in polynomial time. For more details see [35].

3.5.2 Worst case complexity of algorithm from Fig. 3.3

The time complexity of the algorithm is defined by:

1. the complexity of the calculation of the unit concept;
2. the complexity of the calculation of other concepts;
3. the cost for the recalculation of concepts already calculated.

The time expenses mentioned in the second and third point occur during the execution of the procedure `CalculateChildren`. That is why they will be evaluated together.

The complexity of the calculation of the unit element concept for a context (G, M, I) is equal to mn , where $m = |M|$, $n = |G|$. The procedure `CalculateChildren` is executed $L - 1$ times, where $L = |\underline{\mathcal{B}}(G, M, I)|$. (The procedure is used for the calculation of every concept of the concept lattice except the unit element).

The complexity of one execution of `CalculateChildren` is computed as follows. The cycle that calculates the concepts which are specifications of the parent concept (A', A'') is executed once for each attribute from *Active*. The number of such attributes is smaller than or equal to m . The internal cycle that computes the new concept is executed once for every object from the extent of the concept (A', A'') . The size of the extent is bounded by the number of objects in the context n . Other operations (inside the internal cycle and operations in the external cycle located outside the internal cycle of the procedure) are executed at most once during the execution. Their computational complexity is $O(m)$. The complexity of the operation of the intersection of an intent with another intent is also $O(m)$. So, the computational complexity of one execution of `CalculateChildren` is $O(m(nm + m)) = O(m^2(n + 1)) = O(m^2n)$. The general worst-case complexity of the algorithm is $O(m^2nL)$.⁴

The memory requirements of the algorithm are estimated as follows. The maximal level of the recursion in the algorithm is equal to the height⁵ of the lattice. The height of the concept lattice is bounded by $\min(m, n) + 1$. This fact is grounded as follows. In the descending path from the unit element to the zero element of the lattice the cardinality of concept extents monotonically decreases and the cardinality of concept intents monotonically

⁴The L member of the complexity estimate, which denotes the cardinality of the lattice, can be exponential and that is why the general complexity of the algorithm is not polynomial time.

⁵The height of a lattice is equal to the length of the longest descending path in the Hasse diagram of the lattice from the unit element to the bottom element of the lattice.

increases. The minimum cardinality of the concept extent (resp. intent) is zero.

On every recursion step the following information is stored: the currently active parent concept ($O(m+n)$ memory cells are required in order to store the content of extent and intent), the list of attributes that can be used in order to obtain a more specific concept from the currently active one, i.e. the variable *Active* (requiring $O(m)$ memory cells), the space required for storing the content of extent and intent of currently active attributes ($O(m+n)$ memory cells), and the space required for storing the list of attributes for determining already computed concepts, i.e. the variable *AlreadyComputed*. So, the amount of memory required during a single call of the procedure *CalculateChildren* is $O(m+n)$ memory cells. From the estimate and the maximal depth of the recursion during the execution of the algorithm we obtain $O((\min(m, n) + 1) \cdot (m + n))$ as the total amount of required memory for the execution of the algorithm.

3.5.3 Worst case complexity of algorithm from Fig. 3.6

The improvement leading to the algorithm in Fig. 3.6 does not change this time and storage complexity. The reason for this is the following one: the optimization consists in removing from the set *Active* the attributes that are locally subsumed in the current extent by a selected attribute. In the case of a context of the kind (G, G, \neq) there will be no such attributes. That is why the upper bound of the worst case performance of the algorithm is still $O(m^2nL)$ as can easily be seen. The estimate of memory requirements also does not change.

3.6 Experimental comparison of the algorithms

In this section the methodology of comparison of the proposed algorithm with several competing algorithms is described, and then the results of the comparison are reported.

3.6.1 The methodology of comparison

The data sets, on which the comparison was performed, can be divided into two main groups:

- artificially generated datasets
- datasets from real world tasks

Among the artificially generated datasets one can distinguish the following groups:

- deterministic datasets, for which worst case performance occurs
- randomly generated datasets with known parameters of generation

Testing the algorithms on randomly generated data allows to explore the scaling behaviour of the algorithms.

The real world datasets were those listed in Table 4.1. For several big datasets it turned out, that some of the algorithms should be excluded from the comparison due to extremely long runtimes required for them to complete.

We have performed a comparison of the run-time behaviour of the algorithms on two kinds of datasets: on the original ones and on their transposed versions. The reason for testing on the transposed version is the following one. The concept lattice of the transposed context is connected with the concept lattice of the original context as follows: to every concept (A, B) of the lattice of the original context corresponds the concept (B, A) of the lattice of the transposed context, and vice versa. That is why the “complexity”⁶ of these two lattices is the same.

The following algorithms were compared.

1. The algorithm NextClosure, proposed by Bernhard Ganter [32]. We have used the attribute-oriented version with the time complexity $O(m^2nL)$. In the graphs this algorithm is denoted as Ganter.
2. The Nourine-Raynaud algorithm [56].
3. The Norris algorithm [55].
4. The Norris-Godin algorithm. This is the variant of the Norris algorithm proposed by Sergei Obiedkov [47, 57].
5. The bitset-based implementation of the original Charm algorithm [89].
6. The algorithm from Figure 3.3. It is denoted in the graphs as SimpleDFS.
7. The algorithm from Figure 3.6. It is denoted in the graphs as Grail.

⁶That can be defined as $\sum_{(A,B) \in \underline{\mathcal{B}}(G,M,I)} |A| \cdot |B|$

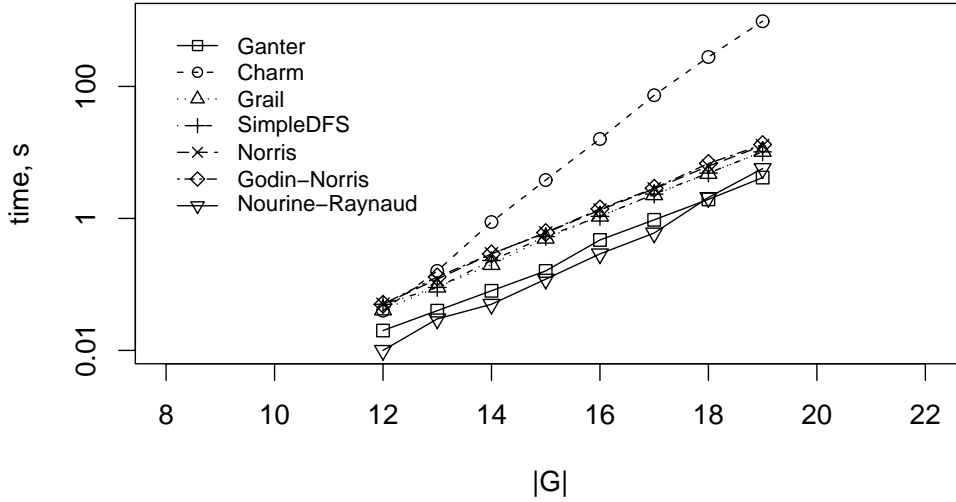


Figure 3.8: Results of the comparison of algorithms in the contexts of the kind (G, G, \neq) . The results are presented using a logarithmic scale.

3.6.2 The results of the comparison for artificial datasets

The results of the comparison in the contexts of the kind (G, G, \neq) are presented in Fig. 3.8. As can be seen from the figure, the algorithms are divided into the three groups. The first group consists of Ganter and Nourine-Raynaud algorithm. The second group consists of two variants of the Norris algorithms, and the algorithms from figures 3.3 and 3.6. The third group consists of the Charm algorithm which has significantly higher runtime compared with other algorithms on the worst-case type of contexts.

The results of the comparisons for sparse contexts are presented in Fig. 3.9. The comparison was performed on the sequence of the randomly generated contexts with a number of objects between 100 and 900, a number of attributes equal to 50 and each object with three randomly distributed attributes. Here the algorithms can be divided into several groups. In the case, in which the number of objects was increasing from 100 to 900 and the number of attributes was fixed, the best behaviour was demonstrated by the Charm and Grail algorithms. The worst in this kind of comparison were the two variants of the Norris algorithm. The interesting effect is that the behaviour of the same algorithm on the lattices built on the basis of the original

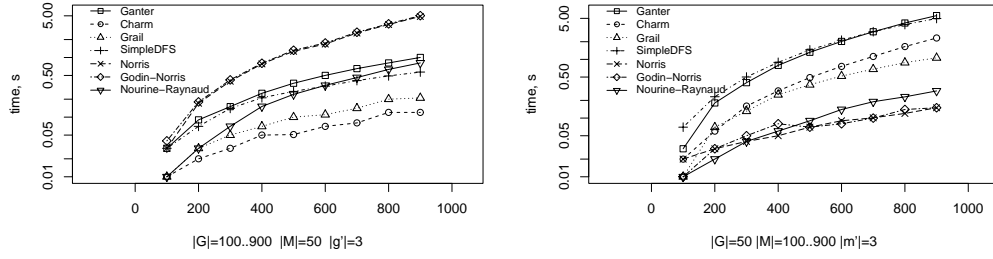


Figure 3.9: Results of the comparison for sparse contexts. See text for more details.

context and of the transposed one is significantly different. The difference is 10 times or more.

In Fig. 3.10 the results of the comparison of algorithms are shown for the randomly generated contexts with a number of objects between 10 and 100, a number of attributes equal to 20, and the fill ratio taking the values 0.3, 0.5 and 0.7. Here one can observe an increase of the runtime of the algorithms with the increase of the density of the contexts which corresponds to the tendency of the increase of the number of concepts in the concept lattice of contexts with higher fill ratio. Another observation is the change of the ranking of the algorithms with the change of the fill ratio. One example of such a behavior can be seen for the example of the Charm algorithm. For the non-transposed contexts with fill ratio equal to 0.3 and to 0.5 the Charm algorithm has the best run-time. With the increase of fill ratio to 0.7 the run time of the algorithm becomes significantly worse.

The examination of the graphs reveals that the optimizations applied to the Grail algorithm really improve its performance. The runtime of the Grail algorithm in all cases is less than the runtime of the SimpleDFS algorithm. At the same time, the performance of the Grail algorithm on the explored contexts is average, and it is outperformed by other algorithms.

3.6.3 The results of the comparison on the real-world datasets

The results of the comparison of the algorithms for the real-world datasets and their transposed variants are presented in Table 3.1.⁷

⁷The characteristics of the datasets for which the comparison was performed can be found in Table 4.1.

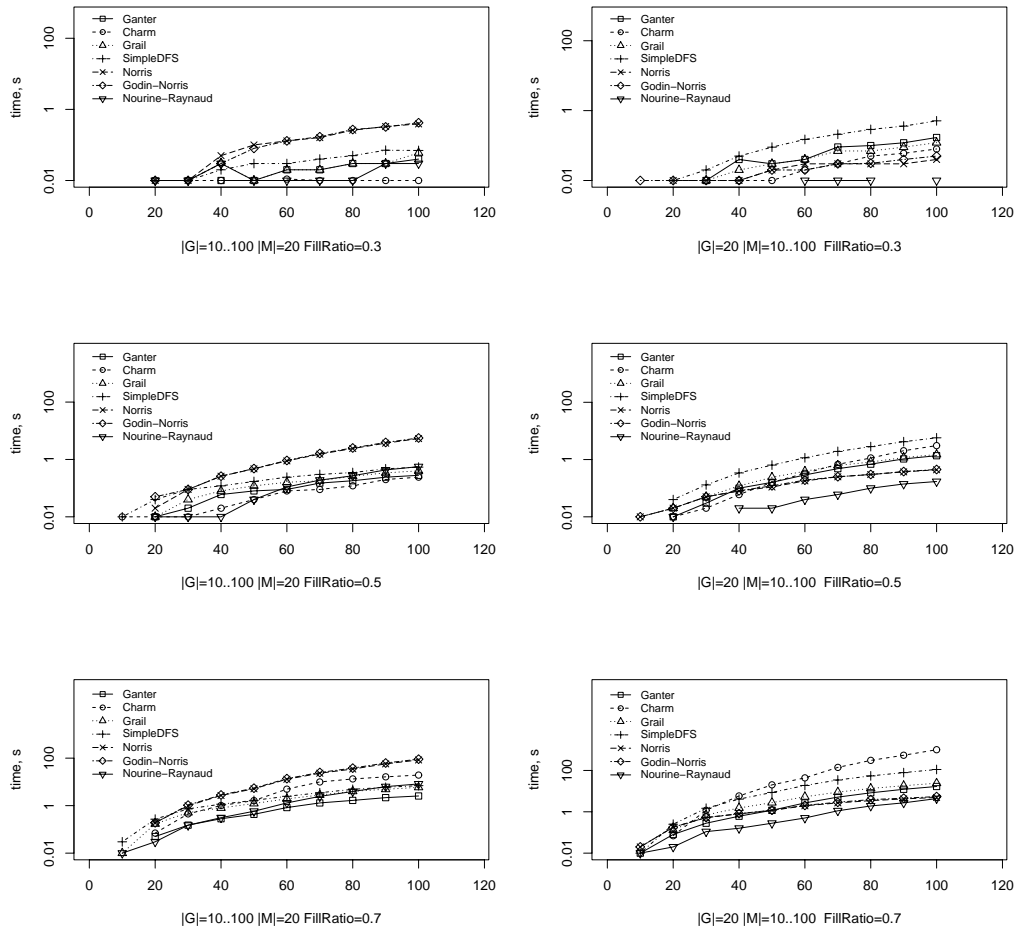


Figure 3.10: Results of the comparison for the contexts with $|G| = 10..100$, $|M| = 20$ and $FillRatio$ taking values 0.3, 0.5, 0.7, and the transposed versions of the same contexts.

Table 3.1: Results of the comparison of the algorithms for the real-world contexts and their transposed variants. The best results are displayed in bold font.

Name	Ganter	Charm	Grail	Nourine-Raynaud
zoo	70	20	30	30
zoo*	61	30	50	10
post-operative	90	30	110	81
post-operative*	331	280	270	40
dbdata0	1201	70	260	521
dbdata0*	2524	551	761	321
primary-tumor	1332	91	200	701
primary-tumor*	2744	1712	1212	421
breast-w	7220	441	2824	7731
breast-w*	29943	10906	6099	5158
breast-cancer	641	150	961	1873
breast-cancer*	5818	6079	3065	571
lymph	1673	230	1182	1452
lymph*	4056	3465	2934	661
spect-all	1172	301	1242	2514
spect-all*	7320	8823	5428	872
MortonRolphRacialStats2	10505	941	2824	13179
MortonRolphRacialStats2*	44394	31455	16424	4487

* denotes the transposed variant of the dataset

	a	b	c	d
1	×			
2		×		
3			×	
4				×

Figure 3.11: Example of a context where all attributes are pairwise disjoint

As can be seen from Table 3.1, on the real-world datasets the best results are achieved by the Charm algorithms, and the Grail is usually second best, on the transposed variants the Nourine-Raynaud algorithm is usually the best, and the Grail is second best.

For the real-world datasets Grail algorithm turned out to be the best one among the FCA-based algorithm in terms of runtime on the original variants of the real-world datasets and turned out to be the second best among all compared algorithm, losing to Charm. The reason for the loss is the efficient bitset-based implementation of the extents intersection operation in Charm.

On the transposed variants of the real-world datasets it turned out to be the second best among all compared algorithms.

3.7 Further ways of improving the algorithm

The next question that arises upon analysing the work of our algorithm is whether it is possible to reduce the number of objects that are checked when calculating the intent and the extent of a new concept by taking into account information that was obtained in previous steps?

In order to clarify potential savings, let us consider the work of the improved variant of the algorithm on the following two contexts:

- A context that has a lot of disjoint attributes (see Fig. 3.11).
- A context in which all attributes form a chain (see Fig. 3.12).

In the first case, after the attribute concept of attribute a was calculated, object 1 cannot be used in further calculations, because it is known that all other attributes are disjoint with attribute a . So, the set of objects used when calculating the extent of the next concept can be shortened. The same applies for all other attributes from this context and so, in summary, the number of operations that check, whether some attribute possesses some particular object, can be shortened by a factor 2 ($\frac{n \cdot (n+1)}{2}$ instead of n^2).

	a	b	c	d
1	×			
2	×	×		
3	×	×	×	
4	×	×	×	×

Figure 3.12: Example of a “chain” context

In the second case, the number of operations performed by the improved algorithm depends on the selected attribute ordering. In the case, when the attributes are selected for computation in an order from a to d , the number of checked objects will be $\frac{n \cdot (n+1)}{2}$; if the order is from d to a , then the number of considered objects will be n^2 . But if, for example, one calculates the first attribute concept for attribute d in the “chain” context, then it is known that all objects contained in the extent of this attribute will also be contained in the extents of all other concepts and thus can be ignored when computing all other extents and then just added to each one after finishing the calculations.

There are several possibilities of improving the performance of the algorithm:

- Using implicit and/or compressed representations of the input data.
- Using more information about the interaction between the attribute used for calculating the new concept from the existing one and other attributes can be used to obtain a more specific concept from a current one.
- Using some attribute ordering strategy in order to shorten the amount of calculation.

In the next section we will consider the first alternative.

3.8 Using implicit representations

One of the ways to improve the performance of algorithms is the usage of implicit representations. This particularly holds when transforming the original representation to an implicit one; performing operations in it and doing the inverse transformation takes less time than performing all operations in initial representation.

In this section we will consider the applicability of one of the implicit representations, namely Binary Decision Diagrams, to the task of the construction of the set of all concepts.

3.8.1 Binary Decision Diagrams

Binary Decision Diagrams (BDD) were introduced at the end of the seventies by Akers [2]. Their wide usage as a data structure for the representation of Boolean functions started in 1986 after the formulation of the set of efficient algorithms for the construction of BDD and performing common logical operations on them [10]. These algorithms were formulated for the subclass of the BDD that is called *Ordered Binary Decision Diagrams* (OBDDs). In the OBDD the order of variables along each path in the diagram is fixed.

They became popular in the logic design community and nowadays are used as one of the main representations of boolean functions in CAD systems for the design of logic circuits. The BDD were applied with great success in tasks of formal model checking, where the algorithms using BDD as the main data structure were used for the verification of systems up to 10^{120} states.

The BDD are based on the decomposition of Boolean function which is usually called “Shannon expansion”. A function f can be decomposed in terms of a variable x as:

$$f = (\bar{x} \wedge f_{\bar{x}}) \vee (x \wedge f_x)$$

The meaning of the “Shannon expansion” is the following one: the boolean function f (or its truth table) dependent from variable x is decomposed into the two functions: f_x and $f_{\bar{x}}$. The function f_x is obtained by replacing in f all occurrences of x by 1 and the function $f_{\bar{x}}$ is obtained by replacing in f all occurrences of x by 0. The Shannon expansion can be read as follows: if x then f_x else $f_{\bar{x}}$.

A Binary Decision Diagram can be viewed as a binary tree, which is transformed into a directed acyclic graph by the elimination of repeated subtrees, and after that applying some node elimination rule (dependent on the type of Decision Diagram) in order to reduce its size further.

In order to represent a boolean function of n variables as an OBDD, some ordering of its input variables should be defined first. Then the boolean function is represented as a directed acyclic graph with two types of nodes: *terminal* and *nonterminal*. The terminal nodes represent function values and the nonterminal ones correspond to function variables.

In the case of boolean functions, there are two terminal nodes: 0 and 1. A nonterminal node corresponds to some input variable x_i and has two children

that are called *low and high successors*. The low successor corresponds to the case when the node variable takes the value 0, and the high successor corresponds to the case when the node variable takes the value 1.

For the classical OBDD the following reduction rules are applied:

1. no two distinct nodes have the same variable index and low and high successors (merging of isomorphic subgraphs),
2. no variable node has identical low and high successors (node elimination).

For solving combinatorial problems dealing with the manipulations of sparse sets another kind of decision diagrams was proposed, namely *Zero-suppressed Binary Decision Diagrams* (ZBDD) [53]. It differs from OBDDs by another node elimination rule. In ZBDDs a node is removed when its low successor points to the terminal 0 node.

The size of a BDD does not directly depend on the number of elements of the truth table on which the function takes 1. BDDs allow to efficiently perform the main boolean operations. The time of performing these operations depends mainly on the size of the BDDs of operands. The efficiency of the BDD-based operations is based on the exploitation of similarities in the structure of boolean functions and the memorization of the results of performed computations.

Many functions widely used in practice have compact representations in BDD. There are boolean functions, for which the size of the BDD varies from linear till exponential in the number of function variables, depending on the selected function variables ordering. Also there exist functions, for which the size of the BDD is always exponential in the number of the input variables [10].

3.8.2 Using a BDD to represent the set of all intents

A Binary Decision Diagram can be used for the representation of the lattice of intents $LI(G, M, I) = \{b \mid (a, b) \in \underline{\mathcal{B}}(G, M, I)\}$ which is isomorphic to the concept lattice $\underline{\mathcal{B}}(G, M, I)$.

In order to achieve this, one can represent the lattice of intents by means of the characteristic boolean function $f : M \rightarrow \{0, 1\}$, where $f(x) = 1 \iff x \in LI(G, M, I)$.

One of the ZBDDs, which corresponds to the concept lattice in Fig. 2.4, is presented in Fig. 3.13. The low successor edges are presented in the picture with dashed lines and the high successor edges are presented with solid ones. The nonterminal nodes are drawn as ovals and the terminal ones as

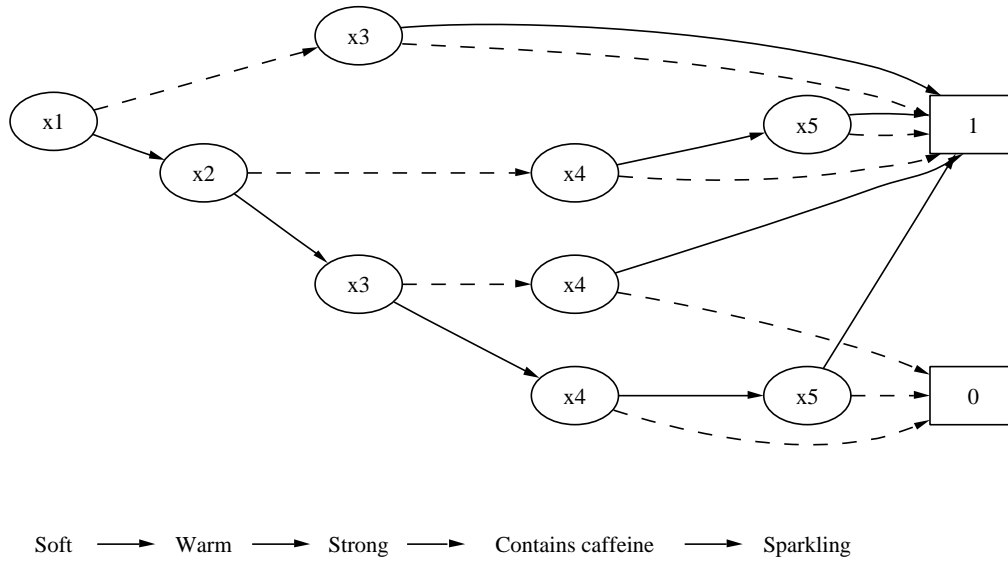


Figure 3.13: The Zero-suppressed Binary Decision Diagram corresponding to the concept lattice in Fig. 2.4

rectangles. The variable names are drawn below the related nodes, for example, the name of the variable x_1 is “Soft” and the name of the variable x_3 is “Strong”.

One of the limitations of this representation is that it is not possible to store the information about the cardinality of the extents in the binary decision diagram of intents lattice. This fact can be demonstrated on the example of boolean lattices.

The boolean lattice of n variables (attributes) $\mathcal{B}(n)$ contains elements corresponding to every possible combination of the variables. For the concept lattice, isomorphic to $\mathcal{B}(n)$, it is possible that for every combination of attributes there is some set of objects corresponding to this combination. Totally, there are 2^n possible combinations of the attributes. So, in order to calculate the size of all extents, one needs to store 2^n different numbers. The BDD corresponding to the boolean lattice $\mathcal{B}(n)$ has exactly one node, i.e. the terminal 1, and the ZBDD has n variables nodes and one terminal node. See Fig. 3.14 for the example of the boolean lattice $\mathcal{B}(3)$ and the corresponding ZBDD.

That is why it is not possible to store in the BDD the information about the cardinality of extents.

The time required to convert the BDD (resp. ZBDD) representation of the intents set to the explicit one is $O(mL)$. If one wants to reconstruct

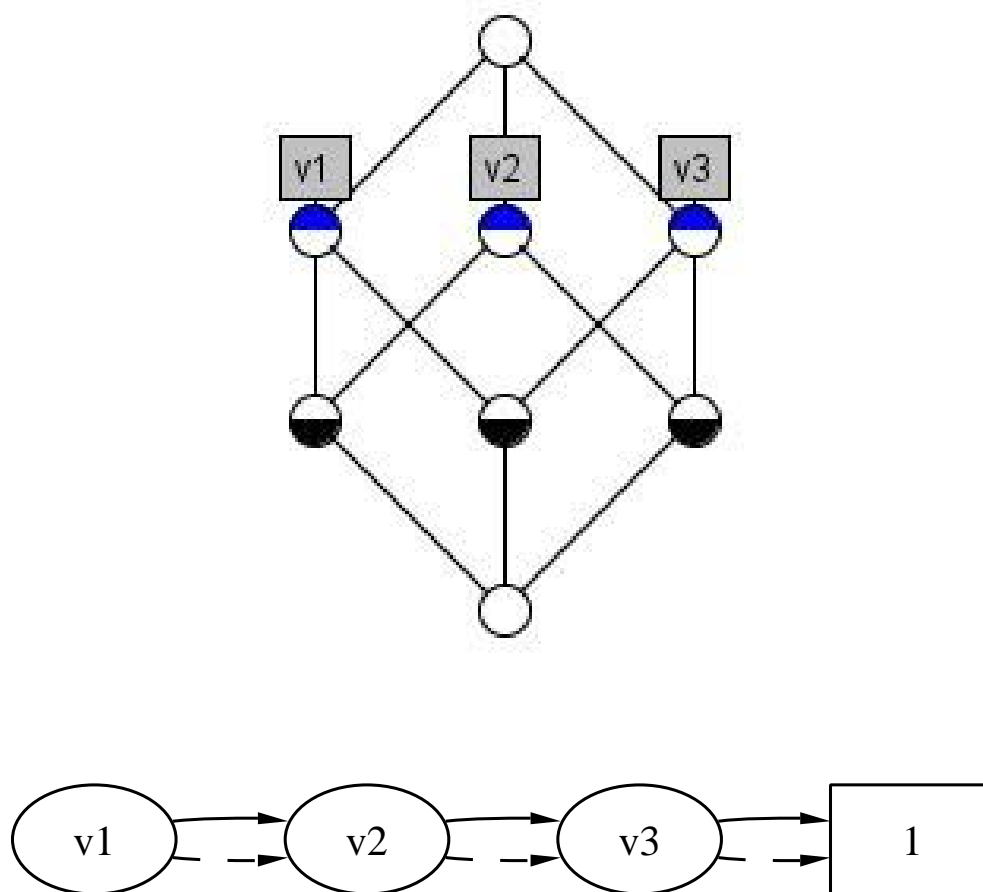


Figure 3.14: The boolean lattice $\mathcal{B}(3)$ and the corresponding ZBDD

the full concept lattice, then this can be achieved in time $O(mnL)$, where $m = |M|$, $n = |G|$, $L = |\mathcal{B}(G, M, I)|$, when the original context is available.

3.8.3 Calculation of all intersections between two sets

Let us reconsider the situation when one wants to calculate the set of all intersections between elements of two different sets.

We will denote as $X \hat{\cap} Y = \{x \cap y \mid x \in X, y \in Y\}$ the operation of the calculation of the set of all intersections between two sets.

Let the following equalities hold: $X = X_0 \cup X_1$ and $Y = Y_0 \cup Y_1$. Then by the distributivity of the set intersection, one obtains

$$X \hat{\cap} Y = (X_0 \cup X_1) \hat{\cap} (Y_0 \cup Y_1) = (X_0 \hat{\cap} Y_0) \cup (X_1 \hat{\cap} Y_0) \cup (X_0 \hat{\cap} Y_1) \cup (X_1 \hat{\cap} Y_1).$$

Now let us consider the case, when the elements of X and Y are the boolean vectors of the same finite length, i.e. they can be represented as:

$$x = x_1 x_2 \dots x_m \text{ where } x_i \in \{0, 1\}, x \in X \text{ or } x \in Y.$$

We will denote by $X_{x_i=k} = \{x \in X \mid x = x_1 x_2 \dots x_{i-1} k x_{i+1} \dots x_m\}$ the subset of X obtained by fixing i -th variable to the value k .

Then for this special case one can specialize the former expansion as follows:

$$X \hat{\cap} Y = 0_1 \wedge \left(\bigcup \left\{ \begin{array}{l} X_{x_1=0} \hat{\cap} Y_{x_1=0} \\ X_{x_1=0} \hat{\cap} Y_{x_1=1} \\ X_{x_1=1} \hat{\cap} Y_{x_1=0} \end{array} \right\} \right) \vee 1_1 \wedge (X_{x_1=1} \hat{\cap} Y_{x_1=1})$$

This expansion means that the result of the computation of the set of all intersections between two sets will include two components in the most general case. The first one of them will be formed as the union of three results of the operation of the computation of the set of all intersections on smaller subsets of the original operands. The first attribute in the selected ordering will be absent in all members of the first component. The second component will include all elements that contain the first attribute. This component will be formed as a result of the operation of calculating the set of all intersections between two subsets of the original operands.

3.8.4 The algorithms using BDD for the construction of the set of all intents

Using the expansion from the previous section one can devise different algorithms for the construction of the set of all intents of the concept lattice.

```

procedure IncrementallyCalculateIntentSet
begin
   $Res = \{M\}$ 
  for each  $x \in X$ 
     $Res := Res \hat{\cap} x$ 
  end

procedure PairwiseIntersectionCalculationOfIntentSet
begin
   $Res = \bigcup \{x\}', x \in G$ 
  do
     $ResPrev = Res$ 
     $Res = Res \hat{\cap} Res$ 
  while  $Res \neq ResPrev$ 
end

```

Figure 3.15: Incremental and pairwise intersection schemes for the calculation of the set of all intents

Two schemes for the calculation set of the set of all intents are presented in Fig. 3.15.

The algorithm `IncrementallyCalculateIntentSet` corresponds to the calculation strategy, which is used in incremental algorithms for the construction of the set of all concepts by Norris [55] and Nourine-Raynaud [56]. In this approach objects are used in the calculation one by one, and for each object all intersections with previously calculated concepts are computed.

The algorithm `PairwiseIntersectionCalculationOfIntentSet` corresponds to the other strategy, when first all intersections between pairs of objects are computed, then all intersection between intersections obtained in the previous step are computed and the computation is going on, until the set of all intersections stabilizes (reaches a fixed point).

This strategy was devised in the hope that the cost of the calculation of all intersections would be low in BDD, and the check for reaching a fixed point can be performed in a fixed time on BDD.

3.8.5 Experimental comparison

We have performed the experimental comparison of the aforementioned algorithms with the modified version of the Norris algorithm. As it was mentioned before, the Norris algorithm can be viewed as the explicit analog of the `IncrementallyCalculateIntentSet` scheme. In one comparison [47] it turned out to be among the best ones, especially for dense contexts. The algorithm was modified in order to produce only the set of all intents.

The schemes, mentioned above, were implemented in JavaTM language.⁸ The implementation of the common BDD algorithms was based on the one described in [3]. We have implemented the variants of the algorithms, which were using both BDD and ZBDD for the representation of the intents lattice. During our experiments it was confirmed that the performance of the ZBDD variant is generally better, except for extremely dense contexts. Here we present the results only for the algorithm using ZBDD-based representation. The tests were performed on the Athlon 1.4 Gz system with 512 MB of RAM running the SuSe Linux operating system. In order to remove the influence of the garbage collection, the garbage collection procedure was called after each experiment.

The early experiments have shown that the performance of the incremental strategy for the computation of the concept set significantly deteriorates in the case of sparse contexts. In order to improve it, we added an additional step to the algorithms. In it the BDD of the elements having nonempty intersection with the object being added to the lattice is selected from the BDD of the lattice. The additional precautions are taken in order not to omit the concept with the empty intent. The performance of this algorithm is slightly worse on dense contexts, but is much better on sparse ones. All results are presented for the variant with the additional step. The experiments also showed, that the incremental scheme in general outperforms the pairwise intersection scheme, so we present results only for the incremental scheme.

In the “classical” algorithms for the BDD the results of computations on intermediate steps are usually stored in hash-tables. This is done in order to reduce the number of performed operations and to prevent the exponential explosion. We have performed the experiments with two variants of algorithms: storing intermediate results in hash-tables and not using hash-tables.

In Fig. 3.16 the results for the contexts of the kind (G, G, \neq) are presented. In this case the size of the concept lattice is maximal w.r.t. the number of

⁸The implementations are quite complicated and have a lot of technical peculiarities. For this reason we do not provide the listings of the algorithms.

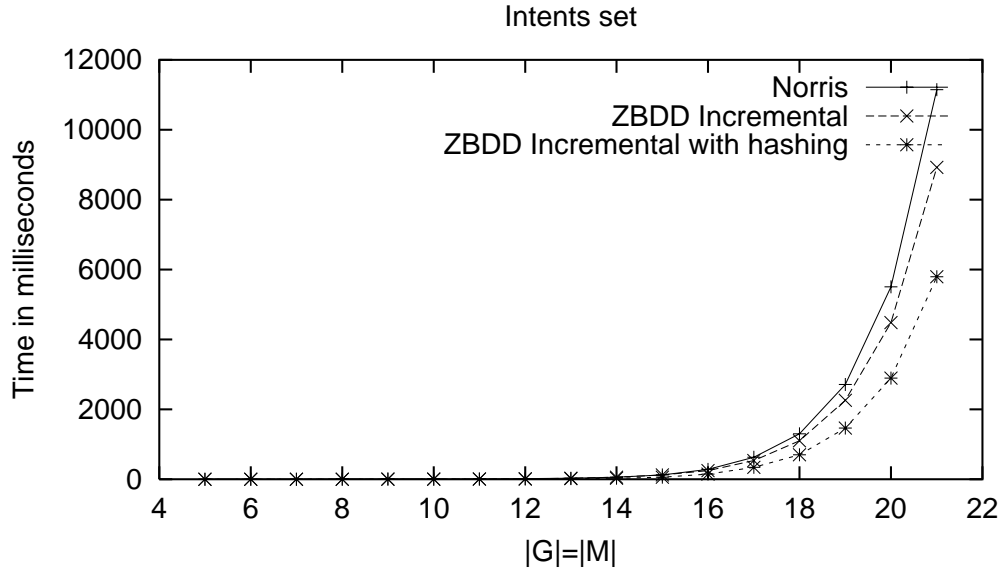


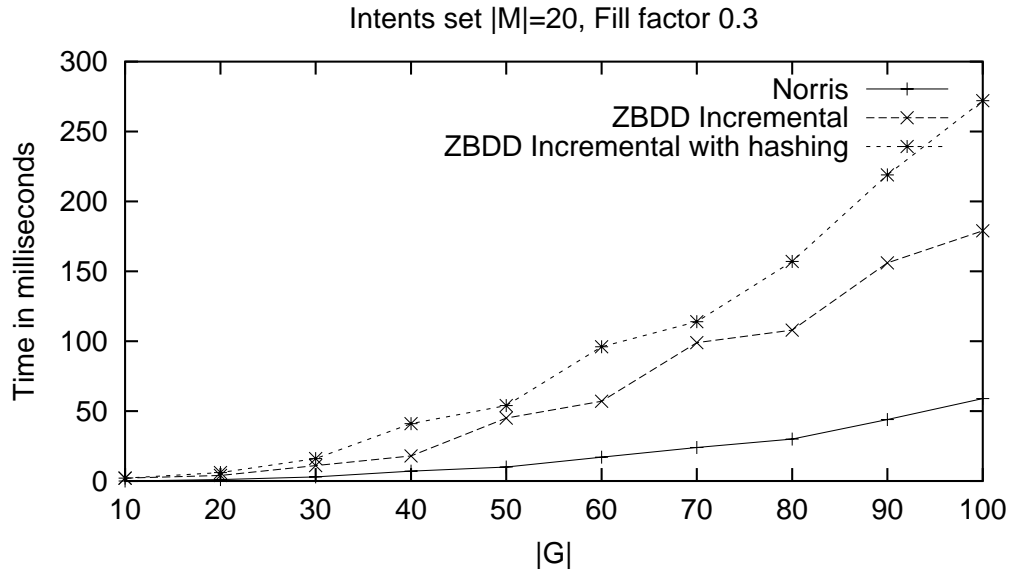
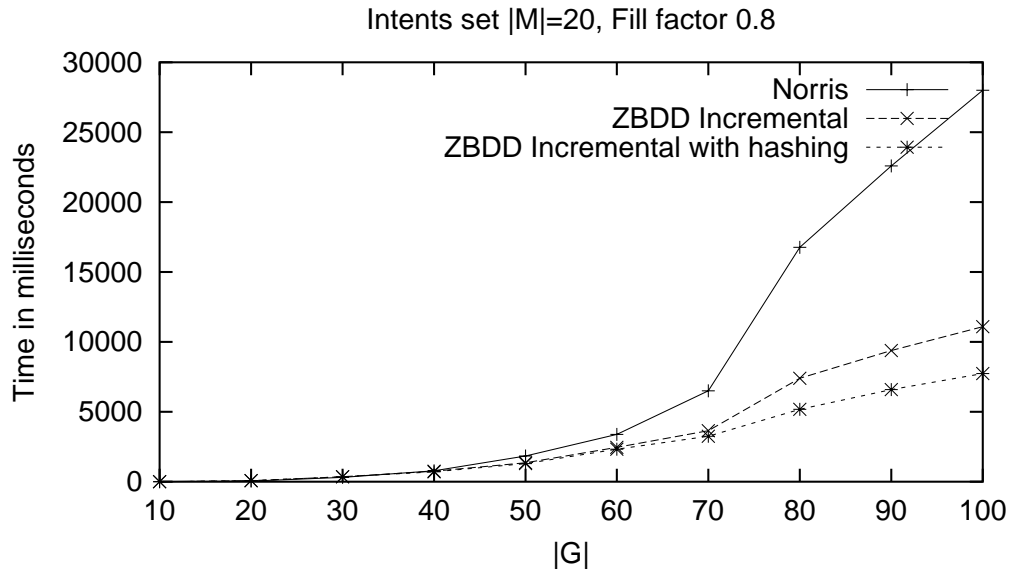
Figure 3.16: Results of the comparison for the contexts of kind (G, G, \neq)

input variables. As it can be seen, the algorithms, based on decision diagrams clearly outperform the Norris algorithm, especially the variant using hash-tables. Actually for such an algorithm the time of the calculation of a decision diagram was never greater than 4 milliseconds and all the other time was taken by the generation of the set of all intents from the decision diagram.

Then we have performed the comparison on the randomly generated contexts with 20 attributes and a number of objects between 10 and 100, with the probability of having a cross in a cell ranging from 0.1 to 0.9. The general results were the following ones: when the probabilities were less or equal to 0.5, then the Norris algorithm was better, while for probabilities above 0.5 the algorithms using the ZBDD were better. The results of the comparison for the cases $P = 0.3$ and $P = 0.8$ are presented in Fig. 3.17 and Fig. 3.18. Another interesting fact was that variants using hash-tables were worse for the case of more sparse contexts.

In order to evaluate the performance in the case of sparse contexts, we have performed the comparison on the contexts with $|M| = 50$, $|G| = 100..900$ and $|g'| = 3$. The results are presented in Fig. 3.19. Here one can see that for the sparse contexts the Norris algorithm clearly outperforms the algorithms using decision diagrams.

We conjecture that these results can be explained as follows. In a dense context there are a lot of similarities between objects and the size of the decision diagram is usually a small one. That is why the intents lattice can be

Figure 3.17: Results of the comparison for $|G| = 10..100$, $|M| = 20$, $P = 0.3$ Figure 3.18: Results of the comparison for $|G| = 10..100$, $|M| = 20$, $P = 0.8$

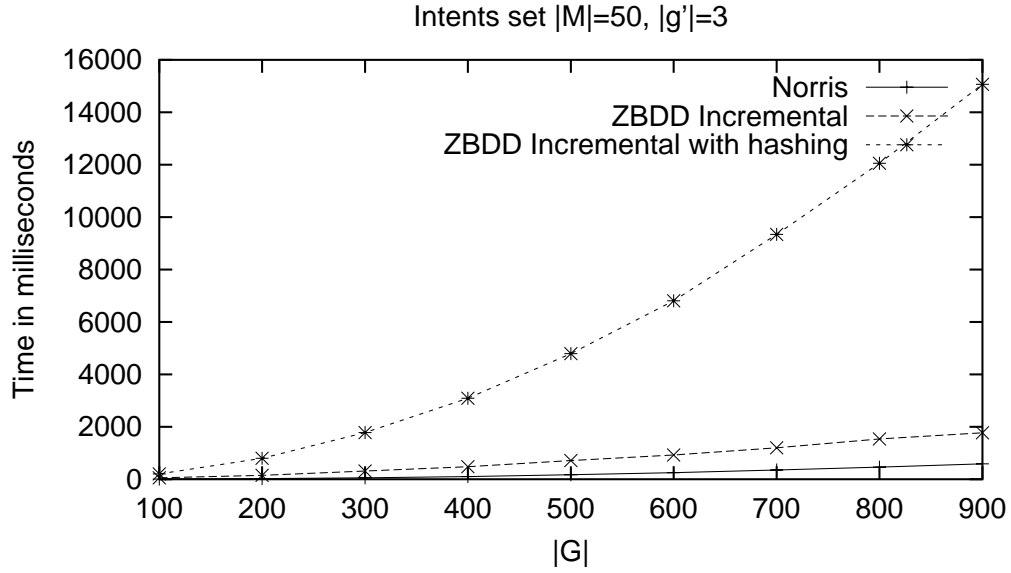


Figure 3.19: Results of the comparison for case $|G| = 50..900$, $|M| = 50$, $|g'| = 3$

represented in a much more compact way than using the explicit representation. But for the sparse contexts the number of similarities is very small and sizes of the implicit and explicit representations differ insignificantly. At the same time, the cost of update of the implicit representation is higher. This leads to the better performance of the explicit representation algorithms on sparse contexts. In a similar way the results about the performance of the algorithms using hash-tables can be explained.

The algorithms using the BDD create a lot of small objects during computations. In JavaTM the cost of object creation is relatively high. That is why we also speculate that in the case of an implementation of these algorithms using the C language the results will be better for them.

3.8.6 Conclusions

The experimental comparison has shown that algorithms for calculating intent lattices using decision diagrams have a very good performance for the case of dense contexts. But in general their performance on the randomly generated contexts cannot be compared with the algorithms using explicit representation. So, these algorithms can be used as additional ones in systems processing concept lattices. Usage of BDD-based algorithms can be recommended in the case, when exponential growth of the number of concepts is

observed during a calculation. Another case, when the usage of BDD-based representation can be profitable is when the system operates with several concept lattices, because some operations on lattices can be performed much faster, using the BDD-based representation.

We have performed experiments using a default ordering of variables. It is well-known that the size of a decision diagram depends on the ordering of variables. So, we conjecture that the performance can be improved in some cases by choosing the appropriate ordering. The task of finding an optimal ordering of a decision diagram is NP-hard. So, one direction of future research can be the development of good heuristics for choosing variable orderings.

Another area, where the usage of BDD can be profitable is the calculation of the base of implications that hold in a context.

3.9 Related work

The amount of algorithms that were proposed for solving the task of computing the set of all concepts (resp. all closed attribute sets) of a binary table is quite big. Some algorithms were proposed before the creation of Formal Concept Analysis [14, 55]. Several algorithms were proposed by the researchers working on the JSM method [88, 45]. One algorithm can be obtained from the algorithm for the construction of knowledge spaces⁹ [21]. Quite naturally, several algorithms were proposed by the researchers using the tools of the Formal Concept Analysis or working in the lattice theory [9, 37, 32, 13, 50, 77, 26, 56]. And, finally, a few algorithms were proposed in the data mining community for solving the task of frequent closed itemsets mining [90, 60, 71].

In the worst case, in the context of the kind (G, G, \neq) , a concept lattice has $2^{|G|}$ nodes. For this case existing algorithms do not differ in performance from the simple “generate-and-test” approach. But their performance differs in other cases. The best algorithms have a complexity that is in some way “linear” in the size of output. A typical complexity is $O(m^2nL)$ or $O(mn^2L)$ where $m = |M|$, $n = |G|$ and $L = |\underline{\mathcal{B}}(G, M, I)|$.

There are several dimensions, along which existing algorithms can be characterised. They are:

1. The way of performing calculations (i.e. batch or incremental).

⁹Knowledge spaces [20] also form lattices. The main difference of knowledge spaces from closure systems is that knowledge spaces are closed under the operation of union, while the closure systems are closed under the operation of intersection of elements.

2. The task being solved by the algorithm (i.e. either generation of the set of all concepts or a more complex task of calculation of the lattice Hasse diagram).
3. The computational complexity of the algorithms and the amount of memory required in computations.
4. The data structures used for the representation of the original data and storing generated concepts (resp. closed itemsets).
5. The mechanism of checking whether the new concept was generated earlier.
6. The technical details of the calculation of new concepts from already existing ones.
7. Batch algorithms additionally can be characterised by the order of generating the elements of the concept lattice.

Now we will describe in more details the mentioned dimensions.

3.9.1 Batch and incremental algorithms

All algorithms can be divided into two big groups by the way of performing calculation, namely batch and incremental ones. The batch algorithms compute a concepts set (resp. a Hasse diagram) once for a whole dataset from scratch. The scheme presented in Fig. 3.1 on p. 22 is a typical representative of such algorithms. The incremental ones produce the concept lattice for the first i objects¹⁰ on the i -th step of the execution of an algorithm. The new concepts set is obtained as a result of the update of the previous one.

Incremental algorithms

As was mentioned above, an incremental algorithm works on the object per object basis. When a new object is added to a context, the following steps should be performed in order to obtain the concept lattice of the updated context from the concept lattice of the context:

1. Generation of new concepts in case when the new object is meet-irreducible.

¹⁰Variants for the attributes are also possible, but at present we are not aware of any.

```

procedure IncrementalConceptSetCalculation
input  ( $G, M, I$ ) // context for which concept set is calculated

 $Concepts := \{(\emptyset, M)\}$ 
for each  $g \in G$ 
    AddObject( $g$ )

procedure AddObject
input   $g$  // the object being added

// Update existing concepts
for each  $Concept \in Concepts, Concept.Intent \subseteq g'$ 
     $Concept.Extent := Concept.Extent \cup \{g\}$ 
if  $\nexists Concept \in Concepts, Concept.Intent = g'$ 
    GenerateNewConcepts(  $Concept \in Concepts, g' \parallel Concept.Intent$  )

```

Figure 3.20: The general scheme of the incremental algorithm for the construction of the set of all concepts

2. Update of concepts already existing in the previous lattice. Only the concepts, the intents of which are included into the new object attribute set, are updated. The new object is added to their extents.
3. Update of links between neighbour concepts in case when the Hasse diagram of the lattice is required.

The typical scheme of the incremental algorithm is presented in Fig. 3.20. The procedure **IncrementalConceptSetCalculation** demonstrates the general scheme of the work of incremental algorithms, i.e. the generation of the concept lattice on an object per object basis. In the procedure **AddObject** the main steps performed by the incremental algorithms are sketched. They are:

- Update of the concepts, the intent of which is included into the new object attribute set g' .
- If the object concept of g is not present in the lattice, then the new object concept will be meet-irreducible (i.e. at least one new concept should be added to the lattice). The generation and addition of new concepts is performed in the procedure **GenerateNewConcepts**. The intents of new concepts can only be obtained either as a new object

intent (i.e. for the object concept) or as a result of the intersecting of the new object intent g' and the intent of some concept that is incomparable (denoted as \parallel in the picture) with g . The technical details of the **GenerateNewConcepts** procedure are different for each incremental algorithm.

In all incremental algorithms for the time being, the new concept is obtained as a result of the intersection of the new object intent g^I and some existing concept (A, B) . The existing concept (A, B) is called a generator concept of the concept $((B \cap g^I)', B \cap g^I)$ [37]. Several generator concepts can exist for one new concept created when updating the lattice to include object g .

For the description of the behaviour of the incremental algorithm the following fact is important: for every new concept that is first obtained when calculating the concept lattice of the context augmented by a new object, there exists a unique maximal (in the cardinality of extent) generator concept and, respectively, a most general (i.e. with minimal intent) generator concept. This fact is important because the new concept extent can be obtained by augmenting the old concept extent with the new object. If a non-maximal concept were used then some object would be missing from the new concept intent.

The details of the incremental algorithms are summarised in Table 3.2.

As was mentioned above, details of the new concepts generation process are different for each algorithm. These details include two (resp. three) components for the algorithms computing a concepts set (resp. a Hasse diagram). They are:

1. The way of calculation of the new concepts.
2. The way of detecting that the new concept is generated from the maximal generator concept.
3. The algorithm for the update of links between neighbour concepts (for algorithms constructing a Hasse diagram).

Several incremental algorithms (i.e. Norris, Nourine-Raynaud, Carpineto-Romano) calculate new concepts as the result of the intersection of a new object attribute set with all concepts of the lattice obtained in the previous step.

The check, whether some generator concept (A, B) is a maximal generator of a new concept (C, D) in the Norris algorithm is performed as follows: all objects that lie in the difference between the set of all objects G_i added till the

Table 3.2: Incremental algorithms for constructing the set of concepts

Algorithm	Solved task	Manner of testing earlier concept generation
Norris [55]	Concept set	List of earlier added objects, checking, whether new concept can be generated, as intersection of earlier added objects.
Godin [37]	Hasse diagram	Search in list of concepts, having intents with cardinality equal to cardinality of new concept intent
Carpineto-Romano [13]	Hasse diagram	The search in Hasse diagram and check whether a new concept is generated from its maximal generator concept
Nourine-Raynaud [56]	Concept set (Hasse diagram)	Lexical tree
AddAtom [26]	Hasse diagram	Ensuring that a concept is generated only once by performing search for the maximal concept generator of the new concept in the Hasse diagram

i -th step and the extent of the generator concept A are checked on inclusion of the new concept intent D in their attribute set. If there is an object g_j , such that $D \subseteq g_j^I$, then the generator concept (A, B) is not a maximal generator concept, because a more general concept $((B \cup \{g_j\})', (B \cup \{g_j\})'')$ is also the generator concept of the (C, D) and its extent contains one additional object g_j .

The worst case complexity of the Norris algorithm is $O(mn^2L)$. The quadratic factor in the number of objects appears mainly due to the complexity of checking the existence of the concept with the new intent that is equal to $O(mn)$. This quadratic factor leads to the impossibility of using the Norris algorithm in tasks characterised by big amounts of objects

The main feature of the Nourine-Raynaud algorithm is the usage of the trie (or lexical tree) data structure for storing the set of concepts. The trie allows to perform the check whether the intersection of a new object attribute set with a concept intent does not occur in the lattice in time $O(m)$. Due to this fact, the complexity of building the concept lattice in this algorithm is reduced to $O(m(m+n)L)$, with space complexity $O((m+n)L)$. At the same time, there is no explicit procedure of the search for the maximal generator concept for a new concept in the algorithm. The algorithm simply considers all possible generators for each concept and forms the extent of the new concept as the union of all extents of generator concepts plus the new object. This procedure is redundant, because some objects can occur in several extents of the generator concepts of the same new concept. In spite of such a redundancy, the Nourine-Raynaud algorithm has the best computational worst-case complexity at present time.

The Godin algorithm was the first algorithm for the incremental update of the Hasse diagram. Two main ideas of this algorithm were the detection and usage of generator concepts in order to calculate a new concept and the use of lists of concepts, in which the concepts with equal cardinality of intents are stored, in order to speed-up the computations. In contrast to the previous algorithms, the Godin algorithm does not necessarily consider intersections of the new object attribute set with all existing concepts. The algorithm in some cases can prune part of the considered concepts when it is known that they would not lead to the generation of new concepts. The other feature of this algorithm is the quite complicated procedure for the calculation of the update of links between neighbour nodes in the Hasse diagram. The worst-case complexity of this algorithm is quadratic in the number of concepts modulo a polynomial factor.

The generalization of this algorithm as the general scheme for the incremental lattice algorithms was proposed in [76]. It is claimed that the computational complexity of this scheme is equal to the computational com-

plexity of Nourine-Raynaud algorithm. Unfortunately, no results concerning an experimental comparison of this algorithm with others have been presented.

The Carpineto-Romano algorithm also solves the task of Hasse diagram construction and update. As in the Norris and Nourine-Raynaud algorithms, the new concepts are calculated by taking the intersection of a new object with every concept of the concept lattice obtained in the previous step. If the intersection does not occur among the intents of the existing concepts, then it is checked, whether the concept used to obtain a new intersection is a maximal generator.

The **AddAtom** algorithm is the most recent addition to the collection of incremental algorithms for the computation of Hasse diagrams. The main difference of this algorithm from the previous ones is the different way of performing a search for generator concepts with which the intersection of a new object would lead to the creation of new concepts. Such an approach allows to remove from consideration a lot of concepts that are not maximal generators of the new concept. The worst-time complexity of the **AddAtom** algorithm is $O(\max\{|g^I| \mid g \in (G, M, I)\}n^2L)$

3.9.2 Batch algorithm

Historically the first algorithm that the author is aware of for the calculation of the set of all concepts was proposed by M. Chein [14]. This algorithm performs the calculation of the set of concepts in a level-wise manner. The calculation starts by calculating the set of all attribute concepts. After that the calculation proceeds by computing all intersections between concepts of the first layer and proceeds in such a way until no new concepts are created as the result of the intersection of the concepts of the previous layer.

One of the most well-known algorithms in the FCA community is the NextClosure algorithm proposed by Bernhard Ganter. It has several discerning features:

- A total lexical order is established on the set of objects or attributes. The order has the following property: if some subset of objects O (resp. attributes) is defined as the next in the order, all other set of objects (resp. attributes) that are subsets of O were already generated earlier. (This order can be called “topological”.)
- The attractive feature of the algorithm is the constant amount of memory used during the work of the algorithm, i.e. $O(n)$ (resp. $O(m)$).

- The order in which the concepts are calculated makes it easy to adapt this algorithm to the task of the calculation of the Duquenne-Guigues base, because all the implications that are used in order to compute the closure of a candidate set are calculated before the candidate set is calculated.
- It is easy to adapt the algorithm to other closure operators.

Also we should mention that the “NextClosure” algorithm serves as a base of the attribute exploration procedure. The Ganter algorithm was developed for solving the task of computing the set of all concepts. The Hasse diagram can also be calculated on the basis of the output of the algorithm in time $O(mn^2L)$.

The first top-down algorithm for the calculation of the Hasse diagram was proposed by J.P. Bordat [9]. The algorithm works in the top-down manner and for each concept calculates the set of all its direct lower neighbours. Then the calculation proceeds with each neighbour. For storing the information about already generated concepts a trie data structure is used.

Another similar algorithm was proposed by C. Lindig [50]. The algorithm of Lindig differs from Bordat by the order of calculations, i.e. the calculations are performed in the bottom-up manner and using a data structure, called a red-black tree, for checking whether the concept is generated for the first time. The worst case complexity of the Lindig algorithm is $O(mn^2L)$.

The “Close by one” algorithm [45] is a bottom-up algorithm that is used for the calculation of the set of concepts. The calculation is performed in the depth-search manner. The new concepts are computed as follows:

- First, the intersection of the current concept intent with some object that lies outside the current concept extent is calculated. This is the new concept intent unless a concept with such an intent was not calculated earlier.
- Then the extent of the current concept is extended to include all other objects that contain all attributes of the new intent among their attributes.

The mechanism of checking the earlier generation of a concept is similar to the one used in the Norris algorithm, i.e. :

- the total order on the object set is defined;
- it is checked whether the new intent is not included in another object that lies outside the current concept extent and is less in the order than the object used to calculate the intent.

The time complexity of the “Close by one” algorithm is $O(mn^2L)$.

3.9.3 Data mining-based algorithms

Several algorithms for mining closed itemsets were developed in the data mining community. They include: Close [58], AClose [59], Titanic [71, 72], Closure [19], MAFIA [11], Closet [61], Closet+ [79], and Charm [89, 90]. The two most efficient among them (judging from the published results) are Closet+ and Charm. It should be noted that in the data mining community most of algorithms were developed for the solution of the association rule mining task, and that is why some of these algorithms generate only the set of intents (also called set of closed itemsets), and determine the number of elements in the extents (i.e. support in data mining terminology).¹¹ Usually the performance of data mining algorithms is compared only experimentally, and in most works theoretical estimates of the complexity of algorithms are not provided.

First we will describe the Charm algorithm. The Charm uses a vertical database layout, in which for each attribute the identifiers of objects, in which the attribute occurs, are stored. Charm performs calculations in depth-first manner. The pseudocode of the Charm algorithm is presented in Fig. 3.21.

The basic¹² Charm algorithm starts its work by transposing the database and calculating the extents for all attributes. (Note, that the pair $(m', \{m\})$, $m \in M$, is not necessarily a concept). After that attribute-extent pairs are put into the queue and the depth first calculation of concepts starts (in the procedure **CharmExtend**). The procedure **CharmExtend** works as follows. While the active queue has elements in it, the first candidate of concept (B', B) is removed from the queue. Then the relationship between the extent of the active candidate and every other candidate of the concept $((C', C)$ in the figure) in the active queue is determined. If $B' = C'$ (i.e. extents are equal) then the closure of the candidates would be the same concept. That is why the second element is removed from further consideration and the elements of the intent of the second element are added to the elements of the first one. If $B' \subseteq C'$, then this means that every concept containing B in its intent should also contain all attributes of C in its intent, and that is why all the elements of C are added to B . If $B' \supseteq C'$ then the concept with extent C' is subsumed by the concept with extent B' and should contain all elements of B . The candidate concept (C', C) is updated to include the elements

¹¹Some of these algorithms, e.g. Titanic and Charm, can be adopted to the task of computing the set of all concepts.

¹²The published algorithm also contains optimizations that speeds up the calculation of closed itemsets that consist of two items.

```

procedure InvertDatabase()
  Concepts :=  $\emptyset$ 
  Queue :=  $\emptyset$ 
  for each  $a \in M$ 
    Queue.add( $(\{a\}', \{a\})$ )
  end
  CharmExtend(Queue)

procedure CharmExtend(Queue)
  for each  $(B', B) \in \text{Queue}$ 
    newQueue :=  $\emptyset$ 
    for each  $f((C', C)) > f((B', B)) \in \text{Queue}$  //elements that are lexically
    //bigger than current one
      if  $(B' = C')$ 
         $B := B \cup C$ 
        remove  $(C', C)$  from Queue
        for each  $(D', D) \in \text{newQueue}$ 
           $D := D \cup C$  //update intents to include the attributes from  $C$ 
        end
      else if  $(B' \subset C')$ 
         $B := B \cup C$ 
        for each  $(D', D) \in \text{newQueue}$ 
           $D := D \cup C$  //update intents to include the elements
        end
      else if  $(B' \supseteq C')$ 
        queue.remove( $(C', C)$ )
        newQueue.add( $(C \cup B, C)$ )
      else //  $B$  and  $C$  are incomparable
        newQueue.add( $(D, B \cup C)$ )
      end //if
    end // for each
    if (!newQueue.isEmpty())
      CharmExtend(newQueue);
    end
    if wasNotComputedEarlier( $(B', B)$ )
      Concepts := Concepts  $\cup \{(B', B)\}$ 
    end
  end // for each

```

Figure 3.21: Pseudocode of the Charm algorithm. For the explanation of details see the text.

of B and moved to the next level candidates queue. When the extents of candidate concepts are incomparable, then their intersection can correspond to the extent of some new concept, which is subsumed by the concept with extent B' . That is why the candidate for the new concept $(B' \cap C', B \cup C)$ is added to the new concept queue.

As the same concept can be generated several times it should be checked whether the new concept is generated for the first time. The check for the first time generation of concept is implemented in Charm by storing all earlier generated concepts in a hash-table. The hash-key is built on the basis of the elements of the concept extent. The elements that have equal hash-keys with a new candidate concept are checked, whether:

- they have equal extent with some already existing concept (variant I);
or
- the intent of the new element is a subset of the intent of some of the existing elements (variant II).

The Charm algorithm can be improved, by moving the test of the previous generation of the concept before the recursive call of the procedure **CharmExtend**. The reason for this is that in case, when the concept was already generated, all the concepts that can be generated from the candidates contained in the *newQueue* were also already generated.

The average case performance of the Charm algorithm was estimated in [89] as $O(lL)$, where l is the average length of extent and L is the size of the lattice. But in the proof of this estimate in the publication, there are some omissions, for example:

- the cost of the calculation of the union of intents is ignored;
- the fact, that some closed sets can be generated several times as a result of intersections of different candidates of concepts and the cost of the further generations is ignored.

The worst-case performance of the Charm algorithm is pretty hard to estimate due to the possible removal of elements from the queue of current concept candidates and to the update of the elements in the queue of concept candidates on the next level. A pessimistic estimate of the complexity of the algorithm is $O(Lm^2(n + m^2))$. It is obtained as follows: the procedure **CharmExtend** is called (except for the first time) after some concept was calculated. The cost of one execution of the procedure is calculated as follows:

- The maximal length of the queue of concept-candidates is equal to the number of attributes, i.e. $O(m)$.

- Each element of the queue is compared with lexically bigger elements of the queue. The number of such comparisons in the worst case is equal to $O(m^2)$. The cost of one comparison is $O(n)$.
- There can be four results of the comparison:
 1. Both candidate concepts have equal extents. Then the candidate intents are merged. The cost of merging intents is $O(m)$. The bigger candidate is removed from future comparisons, that can save about $QL - j$ comparison steps, where QL is the queue length and j is the position of the second from the compared elements in the queue. This saving is ignored in our estimate.
 2. The first candidate concept is a subset of the second candidate concept. Then the intent of the first candidate concept is updated in order to include the intent of the second candidate concept. Also the intents of all candidate concepts contained in the *newQueue* are updated. The time estimate in this case is $O(m + m^2) = O(m^2)$.
 3. The first concept candidate is a superset of the second. The intent of the second candidate concept is updated ($O(m)$) and the second concept is put into the new queue. (There is a saving in the next calculation that is also ignored in our estimate, analogously to the first result).
 4. The second candidate is incomparable with the first one. Then the new concept with an intent that is equal to the union of the first and second candidate intents ($O(m)$) is put into the queue.
- So, the general pessimistic estimate is $O(m^2(n + \max(m, m^2, m, m))L) = O(m^2(n + m^2)L)$.

The memory requirements of the Charm algorithm can be estimated as follows. The memory, consumed during execution of the algorithm, can be divided into:

- the memory, consumed by the algorithm when performing depth search;
- the memory that is used for the storage of already generated concepts in order to check whether the concept with the same intent was already generated.

The number of levels of depth search is limited by the height of lattice, which can be estimated as $O(\min(m, n) + 1)$.¹³ On each level of search the Charm

¹³See p. 36 for the argumentation of this estimate.

algorithm can have at most m unexpanded branches. So, the memory used by the algorithm when performing a search is $O((\min(n, m) + 1) \cdot m \cdot (m + n))$, where the $O(m + n)$ factor is the amount of space required to store a concept. The algorithm also requires space to store all already generated concepts in order to be able to perform the check, whether some concept was already computed. The amount of memory for this task is $O((m + n)L)$. So, the final estimate is equal to $O((m + n)((\min(m, n) + 1) \cdot m + L))$.

The last version of Charm [90] contains one further extension, namely, so-called diffsets that allows in some cases to speed-up calculations and reduce the memory requirements of the algorithm. In this approach, instead of storing the whole extent of the candidate concept, only the difference of the extents between the parent candidate concept (A', A) and the child one $((A \cup \{a\})', A \cup \{a\})$ is stored, i.e. $\text{diffset}(A \cup \{a\}) = A' \setminus A' \cup \{a\}$.

A peculiarity of the Closet+ algorithm is the usage of a special data structure, called FPTree (Frequent Pattern Tree) in order to represent a data table. Basically, FPTree is a trie data structure, built from the data table, with added additional links that allow to quickly calculate the trees of the projected attributes. The FPTree allows in some cases to significantly compress information contained in the context. It should be mentioned that exactly this data structure was used in FPClose*[39] which was the winner of a competition during the Frequent Itemsets Implementation workshop.¹⁴

3.9.4 Other methods

We should mention that a method that lies on the borderline between batch and incremental methods was proposed in [77]. This algorithm allows to build a Hasse diagram of a context (G, M, I) , obtained as the union of two subcontexts (G_1, M_1, I_1) and (G_2, M_2, I_2) , where $G = G_1 \cup G_2$, $M_1 = M_2 = M$ or $G_1 = G_2 = G$, $M = M_1 \cup M_2$ holds, from the Hasse diagrams of the contexts (G_1, M_1, I_1) and (G_2, M_2, I_2) . In [47] it is reported that the algorithm shows a high performance on the worst-case contexts (G, G, \neq) but the performance of the algorithm on other kind of contexts is not very impressive.

3.10 Conclusions

The following contributions were made in this chapter:

- The algorithm Grail for computing the set of all concepts (see Fig. 3.6).

¹⁴<http://fimi.cs.helsinki.fi/>

- The family of algorithms for computing the set of all intents, using binary decision diagrams (see Section 3.8).

The Grail algorithm was compared with other algorithms for the calculation of the set of all concepts. As the comparison criterion the runtime of the algorithms was used.

The comparison of the Grail algorithm with other algorithms for the calculation of the set of all concepts has shown the following:

- The algorithm outperforms the compared FCA-based algorithms (i.e. Ganter, the variants of Norris, Nourine-Raynaud algorithm) in the case of sparse contexts and on real-world datasets.
- But the performance of the algorithm is worse than the performance of the Charm algorithm.
- The algorithm requires $O((\min(m, n) + 1) \cdot (m + n))$ memory space. This is less than the memory requirements of algorithms that require the availability of the set of already calculated concepts for the computations (i.e. all incremental algorithms and some of the batch ones, e.g. Charm). The memory requirements of Grail are bigger than the memory requirements of the Ganter algorithm, but Grail significantly outperforms the Ganter algorithm in most cases, except for dense contexts.

Also the comparison of the Grail algorithm in terms of asymptotic estimates of time complexity and required space were performed. We have shown that the asymptotic time complexity of the algorithm is $O(m^2nH)$. Although the estimated time complexity of the algorithm is worse than the complexity of the Nourine-Raynaud algorithm with best asymptotic complexity, viz. $O(m(m + n)H)$, the run-time of the Grail algorithm on the real-world datasets is in most cases better.

The Grail algorithm can be easily modified for the solution of the following tasks:

- calculation of the Hasse diagram of the lattice;
- calculation of the iceberg concept lattice;
- calculation of the set of concepts of subcontext.

The algorithm was developed by the author during his work on a master thesis [86] in January of 2000.

The second contribution of the chapter is the family of algorithms for the calculation of the set of all intents using BDD for the representation of the set of all intents.

These are the first algorithms that use BDDs for the representation of the set of all intents, i.e. we have proposed this way of representing contexts by means of BDD and have introduced the basic operation required for the computation of the set of all intents. After that we considered alternative strategies for computing the set of all intents.

The empirical comparison of our algorithms using BDDs with other algorithms has shown that these algorithms significantly outperform other algorithms on dense contexts. For the contexts that are considered the worst case for all other algorithms (i.e. the contexts of kind (G, G, \neq)), the calculation of the implicit representation of the set of all intents is exponentially faster, than for usual algorithms!

The BDD-based algorithms were presented for the first time in [87].

We reviewed the related work and pointed out one improvement to the Charm algorithm.

All algorithms that were developed and/or used in comparison in this chapter were implemented using JavaTM language. Their implementations are publicly available at www.sf.net/projects/conexp.

Chapter 4

Search Space Analysis

In this chapter the empirical exploration of several questions, connected with the application of the FCA and related methods in Data Mining, is performed.

In Section 4.1, we describe the characteristics of the real-world and artificial datasets that were used for the experiments. Then the distribution characteristics of the lattices for these datasets are explored in Section 4.2. The results of the exploration lead to the question about the existence of a possible preference between different search strategies employed in the JSM method of hypotheses generation, which is explored in Section 4.3. After that we perform empirical explorations of the effects of concept-lattice preserving context transformations.

4.1 Characteristics of rule mining datasets

Often the testing of new algorithms is performed on randomly generated data. This approach has several advantages:

1. Datasets used in real-world tasks can have a high commercial value and due to this reason are kept confidential. Randomly generated datasets do not have such a limitation.
2. Even when real world datasets are available, their amount can be quite a limited one and a dataset of the required size and with other distribution characteristics can simply be absent. The usage of randomly generated datasets allows to avoid such circumstances. The randomly generated datasets allow to easily explore the scaling behaviour of algorithms.

At the same time, the restriction to only randomly generated datasets can lead to the tuning of the performance of algorithms to perform good on datasets with characteristics that rarely, if at all, occur in practice. There were some experiments in which a significant difference in the distribution characteristics of real-world datasets and randomly generated datasets was found [91].

In this section we will present results of the search space exploration for several real-world and artificial datasets and then consider their implications for the development of algorithms. The characteristics of the analysed datasets are presented in Table 4.1.

Most of the datasets were obtained by performing nominal scaling¹ of the datasets from the UCI machine learning repository [8]. Both vote and vote-both datasets are results of the process of scaling of the same UCI dataset vote, only in the first case the values of voting “yes” were scaled as 1 and “no” and a missing value was scaled as 0. In the second case the standard nominal scaling was used. The spect-all dataset was obtained by merging training and testing datasets of the spect database from UCI machine learning repository. The datasets c73d10 is a PUMS Kansas census dataset. The dataset c20d10 was obtained from it by truncating as to contain only the first 385 attributes. The dbdata0 dataset was obtained from the ARMiner software distribution [18]. The DMC-train and DMC-class are the training and classification datasets that were used at the Data Mining Cup 2003.² The MortonRolphRacialStats2 dataset was obtained from the Pissaro software distribution [62]. The datasets that are marked as artificial were generated using IBM Quest synthetic data generator [64].

The exploration was performed with the help of the algorithm from Fig. 3.6, modified only to store information about the currently generated concept.

We should note that there were several datasets, in which the algorithm for the generation of all concepts was stopped due to the huge number of generated concepts and the impossibility to predict how much time it would require to finish the work. The characteristics of these datasets are given in Table 4.2. We conjecture, that for the datasets audiology and c73d10 the main reason is the high average number of attributes that an object possesses (more than or equal to 70) and for the dataset t20i6d100 the main reason is the high number of objects (99922) combined with a relatively high average number of attributes per object.

¹Nominal scaling is one of the ways to transform the many-valued object-attribute table to the formal context. In it, to each value of the attribute in the original table a new attribute in the scaled context is set in correspondence.

²www.data-mining-cup.de

Table 4.1: The characteristics of the datasets for which a search space exploration was performed. The Fill Ratio is a ratio between the number of cells that contain cross and the total number of cells in the dataset. The Avg. (Max) Attribute describes the average (resp. maximal) number of attributes that an object possesses.

Name	Source	Rows	Columns	Fill Ratio	Avg. (Max) Attributes	Concepts
post-operative	UCI	90	26	0.34	8.97(9)	2378
zoo	UCI	101	28	0.30	9(12)	379
lymph	UCI	148	54	0.25	14(19)	15504
spect.all	UCI	267	23	0.33	8(22)	21550
breast-cancer	UCI	286	43	0.23	10(10)	9918
dbdata0	ARMiner	298	88	0.07	6(21)	2692
primary-tumor	UCI	339	45	0.15	7(13)	3743
voegel6	Pissarro	395	34	0.51	17(30)	153385
vote	UCI	435	18	0.49	9(14)	10644
vote-both	UCI	435	34	0.47	16(17)	247955
MortonRolph-RacialStats2	Pissarro	487	61	0.19	11(22)	32017
breast-w	UCI	699	91	0.11	10(10)	9824
tic-tac-toe	UCI	958	29	0.34	10(10)	59505
flare	UCI	1389	49	0.27	13(13)	28742
c20d10	UCI	2000	386	0.05	20(20)	67195
kr-vs-kp	UCI	3196	42	0.27	11(19)	101121
marketing3	Pissarro	4196	93	0.15	14(14)	872118
dmc2003-train	DMC	8000	834	0.01	8(40)	240967
mushroom	UCI	8416	128	0.18	23(23)	238710
t25i10d10	Artificial	9976	1000	0.02	25(63)	3853929
dmc2003-class	DMC	11177	832	0.01	7(38)	384252
letter	UCI	20000	282	0.06	17(17)	12018941
t10i4d100	Artificial	98395	1000	0.01	10(28)	2405732

Table 4.2: The characteristics of the datasets with a “too big” search space for the enumeration of all concepts

Name	Source	Rows	Columns	Fill Ratio	Avg. (Max) Attributes	Concepts
audiology	UCI	226	178	0.385	70(70)	$> 415 \cdot 10^6$
c73d10	UCI	10000	2178	0.034	73(73)	$> 35 \cdot 10^6$
t20i6d100	Artificial	99922	1000	0.020	19.90(47)	$> 22.3 \cdot 10^6$

4.2 The distribution characteristics of lattices

For the aforementioned datasets we have explored the distribution of the number of concepts depending on the extent and intent size.

Let a concept $(A, B) \in \underline{\mathcal{B}}(G, M, I)$ be given. Then the *relative extent size* is defined as $\frac{|A|}{|G|}$, where G is the set of all objects of context. The values of relative extent size lie in the $[0, 1]$ range. The *relative frequency* of the number of concepts for the extent (resp. intent) size s is the value $\frac{|\{(A, B) \mid |A|=s\}|}{|\underline{\mathcal{B}}(G, M, I)|}$ (resp. $\frac{|\{(A, B) \mid |B|=s\}|}{|\underline{\mathcal{B}}(G, M, I)|}$).

The distribution characteristics of the distribution of the number of concepts in dependence of the extent size are presented in Table 4.3. The graphs of the relative frequency function of the distribution of the number of concepts in dependence of the relative extent size and of the relative frequency function of the distribution of the number of concepts in dependence of the intent size are presented in Fig. 4.1 and Fig. 4.2, respectively.

On the basis of the characteristics presented in the table and of the graphs of the distributions in Fig. 4.1, it is possible to draw the following conclusions, assuming that the explored datasets are typical:

1. In distributions of the size of extents for most explored datasets exists a narrow area, where the majority of all concepts are located. (The notable exception is the voegel6 dataset.) The distribution is usually unimodal and has a mode³ close to zero.
2. Concepts in the aforementioned narrow area usually have small sizes of extents. In 20 from 23 explored datasets, a half of the concepts were

³Mode of a random variable X , X_{mode} is the most likely value of X . For the discrete random variable X the mode of X is such a value x for which $P(X = x)$ is larger then for any other value.

Table 4.3: The characteristics of the distribution of the concept count in dependence of the extent size of the concepts of the explored datasets. The columns with names Q(Value) describe the absolute and the relative values of the corresponding quantiles of the distribution.

Name	Q(0.5)	Q(0.9)	Q(0.95)	Q(0.99)
zoo	10(9.9%)	35(34.7%)	42(41.6%)	75(74.3%)
vote	12(2.8%)	49(11.3%)	74(17.0%)	140(32.2%)
vote-both	16(3.7%)	58(13.3%)	82(18.9%)	113(26.0%)
voegel6	49(12.4%)	130(32.9%)	162(41.0%)	224(56.7%)
tic-tac-toe	6(0.6%)	21(2.2%)	31(3.2%)	72(7.5%)
t25i10d10	3(0.0%)	9(0.1%)	18(0.2%)	54(0.5%)
t10i4d100	3(0.0%)	12(0.0%)	19(0.0%)	119(0.1%)
spect-all	10(3.8%)	20(7.5%)	26(9.7%)	42(15.7%)
primary-tumor	5(1.5%)	16(4.7%)	25(7.4%)	57(16.8%)
post-operative	6(6.7%)	16(17.8%)	23(25.6%)	42(46.7%)
mushroom	16(0.2%)	236(2.8%)	444(5.3%)	1216(14.5%)
MortonRolph-RacialStats2	6(1.2%)	18(3.7%)	26(5.3%)	60(12.3%)
marketing3	8(0.2%)	37(0.9%)	60(1.4%)	148(3.5%)
lymph	6(4.1%)	16(10.8%)	22(14.9%)	42(28.4%)
letter	5(0.0%)	17(0.1%)	26(0.1%)	63(0.3%)
kr-vs-kp	22(0.7%)	106(3.3%)	167(5.2%)	383(12.0%)
flare	18(1.3%)	146(10.5%)	247(17.8%)	652(46.9%)
dmc2003-train	6(0.1%)	20(0.3%)	31(0.4%)	75(0.9%)
dmc2003-class	6(0.1%)	23(0.2%)	33(0.3%)	77(0.7%)
dbdata0	5(1.7%)	19(6.4%)	28(9.4%)	61(20.5%)
c20d10	16(0.8%)	87(4.4%)	154(7.7%)	446(22.3%)
breast-w	4(0.6%)	48(6.9%)	94(13.5%)	288(41.2%)
breast-cancer	6(2.1%)	21(7.3%)	33(11.5%)	75(26.2%)

Figure 4.1: The relative frequency function of the distribution of the extent sizes of explored datasets. The size of the dataset is shown in the relative scale.

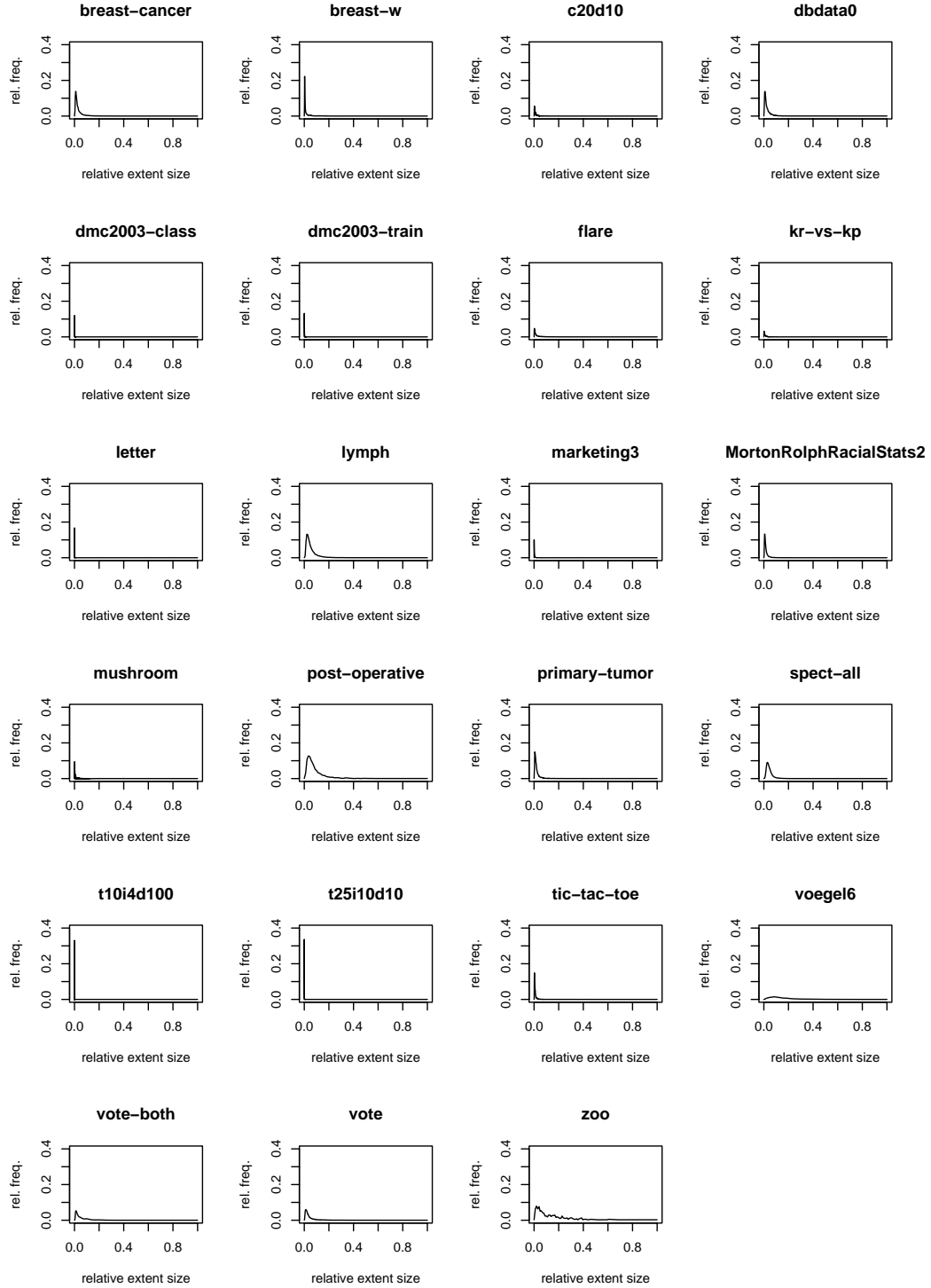
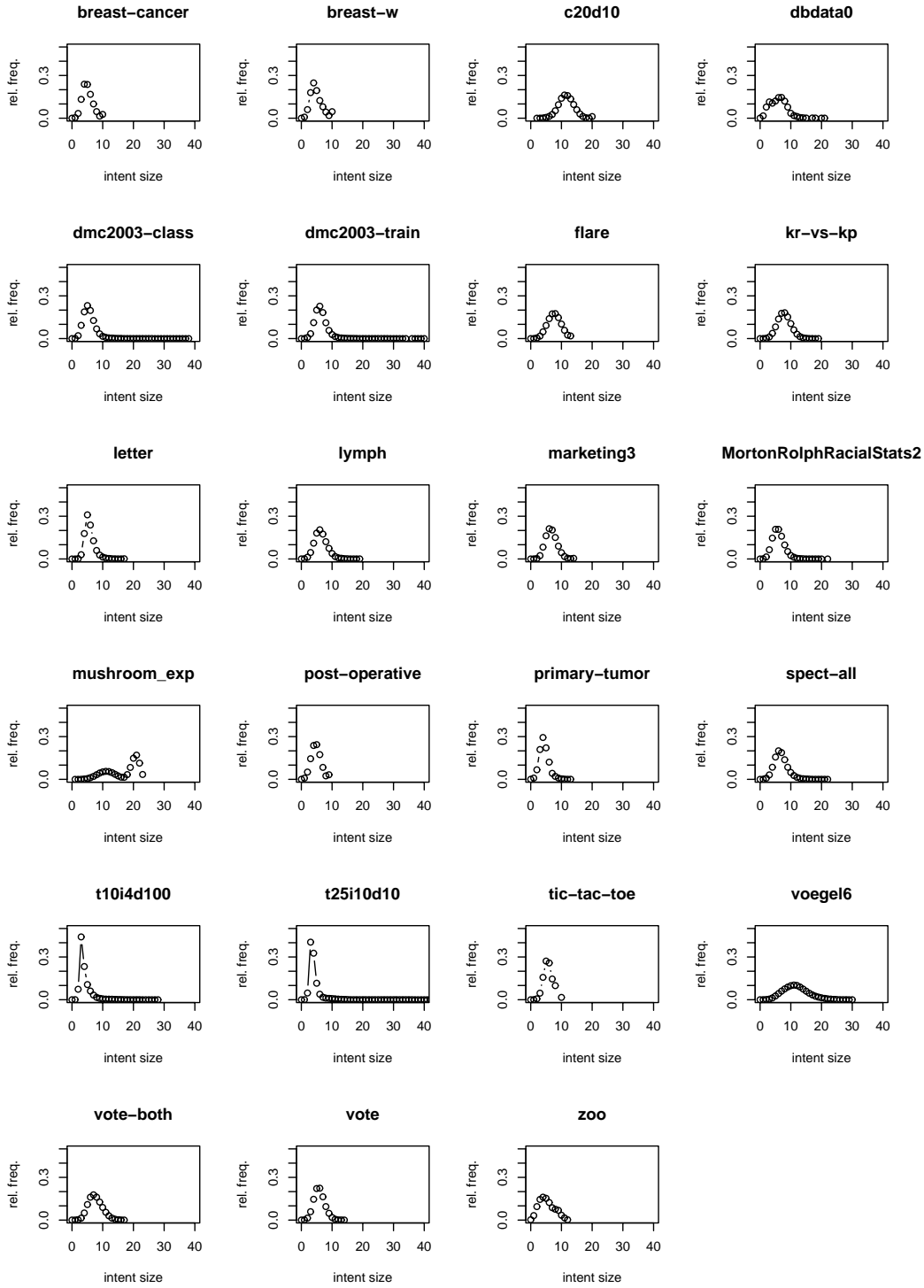


Figure 4.2: The relative frequency function of the distribution of the intent sizes of the explored datasets



having extents with a size less than or equal to 16. For the other 3 the median is located at 18, 22 and 49 objects.

3. One can also observe the difference in the distributions of the artificially generated and real-world datasets. For example, the maximum of the relative frequency function⁴ of the artificial t10i4d100 and t25i10d10 datasets is higher than 0.3 of the number of all concepts, and for other datasets the maximum of the relative frequency function is in one case somewhat bigger than 0.2 (breast-w) and in all other cases is smaller than 0.2 of the number of all concepts.⁵

The distributions of the size of intent are more uniform and mostly unimodal. The modes of the distribution for most datasets are located in the range from 3 to 7 objects in extent. The non-typical datasets from this viewpoint are c20d10 (the position of the mode is 9 objects in the extent), mushrooms (20 objects in the extent for the mode) and voegel6 (11 objects). For the real world datasets the value of the mode of the distribution of the number of concepts by the size of the intent is usually located in the range from 0.10 till 0.31. The artificially generated datasets have higher values of mode (viz. 0.40 and 0.44) of the total number of concepts.

These observations lead to the following consequences.

- There is a significant difference in the characteristics of artificially generated and real-world datasets.
- It is worth to perform a special optimization of algorithms in order to attain a good performance when generating concepts with small extent sizes. Such an optimization can be (but is not limited to):
 - Using two versions of the same algorithm that employ different data structures for the calculation of concepts with a large resp. small size of extents (i.e. using linked lists for storing indices of

⁴The maximum of the relative frequency function in this case characterizes the maximal amount of concepts with equal size of extents, which occur in lattice.

⁵This difference is of practical significance when testing the algorithms for computing iceberg concept lattices. For the datasets with the distribution of number of concept relative to extent size similar to the distribution of artificially generated datasets, the increase of the number of concepts when lowering the minimal interesting size of the concept extent (i.e. support) will be significantly higher than for the datasets with distribution similar to the distribution of the explored real-world datasets. This can lead to the situation when algorithm that performs well on the real-world datasets will run out of memory on the artificial dataset. (This is especially likely for an algorithm computing concept lattice in a level-wise manner.)

- objects when computing the concepts with large extents resp. arrays for storing indices of objects when computing the concepts with small extents).
- Building a “projected” subcontext containing only the objects of the currently computed extent and the attributes related to the objects, once the size of the extent is smaller than a cutoff value. Further computations are performed on this subcontext. We suggest that the cutoff value should lie in the range between 10 and 16 objects in extent.
- On the basis of the shape of the distribution of the number of concepts in dependence of the size of the extent, one can hypothesize that for the JSM method of automatic hypothesis generation for real-world datasets the top-down search strategy is much more preferable. We will check this hypothesis in the next section.

4.3 Exploration of binary datasets for JSM

In this subsection we will check, whether the hypothesis concerning a preference of the top-down calculation strategy for the JSM-method applied to the real-world datasets is confirmed by empirical data.

First, we will provide some basic information about the JSM-method.

4.3.1 Short information about the JSM-method

The JSM-method for hypothesis generation was proposed in the group of Prof. V.K. Finn working in VINITI in Moscow [25]. The name of the method is the abbreviation of the initials of the famous English philosopher John Stuart Mill.

The method was created for finding dependencies in domains in which cause-effect dependencies are mostly strictly deterministic.⁶ The feature that distinguishes the JSM method from many other machine learning methods is that the classification of an unknown example can take one of four values:⁷

⁶One example of such a domain is the Structure-Activity Relationship Problem, a prediction of the presence/absence of certain biological activities on the basis of presence/absence of certain structural subcompounds of biochemical compounds.

⁷Typical machine learning methods for two-class problems provide either two-valued classification, always making a decision whether an unclassified example possesses or not the target property, or three-valued classification, when additionally the “no prediction” decision can be done.

“+” the example is classified as possessing the predicted property;

“−” the example is classified as not possessing the predicted property;

CON there is an evidence both in favour of possessing the predicted property and in favour of not possessing the predicted property;

τ there is no evidence to make any conclusion.

There are several modifications of the JSM method. The most well-known among them are:

- simple JSM method [44];
- generalized JSM method [24];
- quantitative JSM method [40].

The general scheme of JSM-reasoning has the following steps:

1. Gathering and preparation of training data. The training data consist of positive (“+”-examples) and negative examples (“−”-examples) which are used for concept learning. Additionally a set of undetermined examples can be provided, for which it is not known whether they possess the goal property or not.
2. Induction of candidates for the hypothesis. Candidates are generated from positive and negative examples. How the candidates are generated differs depending on the method.
3. The hypothesis falsification. Among the generated candidates of hypothesis only those that do not satisfy all falsification criteria are selected.
4. The hypothesis candidates that pass the previous step are used for the classification of the undetermined examples.
5. Check, whether there are any undetermined examples left?
6. Repetition of the steps 2–4 in case that some of the unclassified examples were classified and added to the training set, and there are some unclassified examples left.
7. Check of the completeness of the set of non-falsified hypotheses. Completeness is understood in the sense that the set of hypotheses explains all examples in the training data.

In the JSM-method the details of the object representation are not specified. The requirement that analyzed objects should satisfy is the availability of the computable intersection operation on objects. The result of the intersection operation of two or more objects is a fragment.⁸ The main operation, defined for a fragment, is the inclusion operation that tests, whether the fragment is included in an object. The set of fragments forms a semilattice.

In the simple JSM method only the elements of fragments of the semilattice formed by objects of “+”-examples (resp. “-”-examples) are considered as candidates for hypotheses.

The intersection operation on the set of fragments should satisfy the following criteria:

idempotency: $\forall f_i \in \mathcal{F} : f_i \cap f_i = f_i$.

reflexivity: $\forall f_i, f_j \in \mathcal{F} : f_i \cap f_j = f_j \cap f_i$

associativity: $\forall f_i, f_j, f_k \in \mathcal{F} : (f_i \cap f_j) \cap f_k = f_i \cap (f_j \cap f_k)$

existence of zero: $\exists! s \in \mathcal{F} \forall f_i \in \mathcal{F} : f_i \cap s = s$

The notation f_i, f_j is used to denote fragments, \mathcal{F} denotes the set of fragments of the semilattice and s the empty fragment.

For the falsification of hypothesis candidates traditionally the counterexample inhibition criterion is used. According to this criterion, the “+”-hypothesis candidate is not falsified if there exists an object from the “-”-examples such that the “+”-hypothesis candidate is included into the object. The same criterion applies also for negative examples.

The set of “+”-hypothesis candidates forms a finite semilattice. The same is true for the set of “-”-hypothesis candidates. The semilattice is isomorphic to some concept lattice when additionally the \perp element is added to the semilattice. The elements of the semilattice can be divided into hypotheses and non-hypotheses.

We will denote the set of hypotheses as *Hyp* and set of non-hypotheses as *NonHyp*. The counterexample inhibition criterion possesses the property of antimonotonicity, i.e. if some fragment is falsified according to the criterion, then all fragments that are included in this fragment will be also falsified. That is why in case of the application of the counterexample inhibition criterion among hypotheses (i.e. *Hyp*) the set of hypotheses which are minimal by inclusion can be identified (this set will be called a *positive border* and denoted as *HypBorder*) and among non-hypotheses (i.e. *NonHyp*) the set of

⁸An object can be seen as equivalent to the maximal fragment that is included in object. The set of objects \mathcal{O} is included in the set of fragments \mathcal{F} , i.e. $\mathcal{O} \subset \mathcal{F}$.

non-hypotheses which are maximal by inclusion can be identified (resp. called *negative border*, which will be denoted as *NonHypBorder*).

4.3.2 Description of experiments

The task of search space exploration was to obtain empirical information in order to answer the question which calculation strategy is preferably used in algorithms implementing the JSM method, namely top-down or bottom-up.

The preference of one strategy to the other was determined based on the following assumptions:

- When calculating the minimal hypotheses using the top-down strategy then all elements of the non-hypotheses set in the lattice (i.e. *NonHyp*) are calculated and also all minimal hypotheses (i.e. *HypBorder*). So, the computational complexity of the top-down strategy is assumed to be $O((NonHyp + HypBorder) \cdot HypCheck)$. The *HypCheck* denotes the computational complexity of the procedure that determines whether a fragment is a hypothesis or not.
- When calculating the minimal hypotheses using the bottom-up strategy then all hypotheses part of the lattice should be calculated (i.e. *Hyp*) and also all elements of the set of maximal by inclusion non-hypotheses (i.e. *NonHypBorder*). So, the computational complexity of the bottom-up strategy is assumed to be $O((Hyp + NonHypBorder) \cdot HypCheck)$.

The information about the cardinality of the sets of hypotheses and non-hypotheses for two-class datasets is presented in Table 4.4 and in Table 4.5. In these tables the information about the values *Hyp*, *NonHyp*, *HypBorder* and *NonHypBorder* for the eight two-class datasets is provided. For the general dimension-based characteristics of the datasets see Table 4.1.

As can be seen from Table 4.6, in four cases the top-down strategy was preferable (the ratio ranges from 0.03 till 0.37), in three cases the bottom-up calculation strategy was preferable (the ratio in range 1.80 to 2.34), and in one case the strategies were almost equal (the ratio equals 0.97). So, the empirical data do not support the validity of the hypothesis about a preference of the top-down calculation strategy over the bottom-up one.

4.4 Effects of context transformations

As was mentioned in Chapter 2, there are two context transformations that do not change the structure of the concept lattice, namely the clarification

Table 4.4: Cardinalities of the set of hypotheses, the set of non-hypotheses, of minimal by inclusion hypotheses (denoted as “pos border”) and of maximal non-hypotheses (denoted as “neg border”) for the “+”-class.

Name	# of hypotheses	# non hypotheses	# pos border	# neg border
spect.all	17315	1595	170	148
dmc2003.train	228983	2820	3750	807
mushroom	85960	7403	109	197
vote	368	570	14	32
breast-cancer	336	1625	108	346
tic-tac-toe	11635	10004	553	1438
kr-vs-kp	7943	14944	137	480
vote-both	16200	12408	127	328

Table 4.5: Cardinalities of the set of hypotheses, the set of non-hypotheses, of minimal by inclusion hypotheses (denoted as “pos border”) and of maximal non-hypotheses (denoted as “neg border”) for the “-”-class.

Name	# of hypotheses	# non hypotheses	# pos border	# neg border
spect.all	1	91	1	14
dmc2003.train	3256	900	503	432
mushroom	82941	3607	61	120
vote	1222	3060	28	32
breast-cancer	1144	2268	207	512
tic-tac-toe	4401	6788	210	716
kr-vs-kp	9222	28171	301	996
vote-both	82170	23653	319	687

Table 4.6: Results of the comparison of the number of explored elements for top-down and bottom-up calculation strategies. Ratio is equal to $\frac{|TopDown|}{|BottomUp|}$, where $|TopDown|$ is the number of elements explored by the top-down calculation strategy and $|BottomUp|$ is the number of elements explored by the bottom-up strategy.

Name	Top-down strategy	Bottom-up strategy	Ratio
spect.all	1857	17478	0.11
dmc2003.train	7973	233478	0.03
mushroom	11180	169218	0.07
vote	3672	1654	2.22
breast-cancer	4208	2338	1.80
tic-tac-toe	17555	18190	0.97
kr-vs-kp	43553	18641	2.34
vote-both	36507	99385	0.37

and the reduction of a context. Here we will explore, whether it is worth to perform such transformations as a preprocessing step, when constructing concept lattices. The results of preprocessing a number of contexts from the datasets in Table 4.1 are presented in Table 4.7.

There were 23 analysed datasets. Among them 3 are artificial and 20 are real-world ones. The real-world datasets can be divided into the datasets that are originally binary (i.e. all attributes of such datasets are binary from the very beginning) and ones that are obtained by the many-valued nominal scaling. There were 4 originally binary datasets and 16 datasets obtained by nominal scaling.

From the 23 datasets, the context clarification transformation lead to a reduction in 11 cases. The impact of this reduction was quite a significant one (more than 30 percent of attributes removed) in three cases (the precise figures are 59%, 36% and 35%). One of these cases belongs to the artificial datasets, and two others are originally binary. In both cases, the main part of the reduction was the elimination of the empty attributes from the dataset. There were five cases, in which the impact of the reduction was medium (in the range of 7–16 percent). Among the cases, two datasets were artificially generated, one was originally binary and two were obtained by the scaling from the object-attribute-value format. In three other cases the effect of the selection of distinct attributes was insignificant.

Table 4.7: Results of applying clarification and reduction transformations to the datasets

Name	Rows	Dist. objects	Irred. objects	Cols.	Dist. attr.	Irred. attr.
post-operative	90	80	80	26	26	25
zoo	101	59	44	28	26	22
lymph	148	148	148	54	53	52
spect.all ²	267	228	133	23	23	23
breast-cancer	286	272	272	43	43	42
dbdata0 ²	298	258	193	88	79	61
primary-tumor	339	309	275	45	45	43
voegel6	395	378	261	34	34	34
vote	435	298	167	18	18	18
vote-both	435	342	315	34	34	34
MortonRolphRacialStats2	487	477	456	61	61	60
breast-w	699	463	461	91	91	91
tic-tac-toe	958	958	958	29	29	29
flare	1389	527	527	49	48	45
c20d10	2000	783	783	386	159	143
kr-vs-kp	3196	3196	1443	42	42	41
marketing3	4196	4000	4000	93	93	92
dmc2003-train ²	8000	4151	3535	834	537	462
mushroom	8416	8124	8124	128	114	100
t25i10d10 ¹	9976	9657	9112	1000	929	891
dmc2003-class ²	11177	5547	4643	832	540	474
letter	20000	18668	18668	282	281	278
t10i4d100 ¹	98395	86454	80860	1000	845	840

¹ dataset is artificial² dataset is originally binary

The application of context attribute transformation leads to a reduction of the number of attributes for the explored contexts in 17 cases. In all cases, in which an attribute clarification leads to an elimination of some attributes from the dataset, for the datasets used in the experiment, the attribute reduction procedure leads to the elimination of additional attributes. In 10 cases out of 17, the number of additionally eliminated attributes was greater than what attribute reduction achieved through the attribute clarification. In five cases the number of attributes additionally eliminated by the attribute reduction transformation was smaller than the number of attributes eliminated by attribute clarification. The number of additionally eliminated attributes was at most 20% of the attributes in the original dataset. Among the datasets, for which the attribute reduction transformation eliminated additionally more than 10 attributes, were three artificial datasets, two originally binary datasets and one obtained by nominal scaling. Also, the additional effect of attribute reduction on such datasets was usually smaller than the effect of the attribute clarification transformation.

In summary, the results indicate that it is worth to perform attribute clarification and attribute reduction transformations as one of the data preprocessing steps, in the initial stage of data preprocessing. If other kinds of data preprocessing are applied, such as feature subset selection, then it can be expected that the effect of the application of transformations will be small. The attribute clarification and attribute reduction preprocessing operations seem to have stronger effects on the originally binary data. One plausible explanation for the poor performance of these two transformations on datasets obtained by nominal scaling could be the fact, that the datasets from the UCI machine learning repository are in most cases already preprocessed so that duplicated attributes were already eliminated.

4.5 Conclusions

In the chapter an exploration of the search space characteristics of the data mining datasets was performed.

The results of the exploration showed that there is a significant difference in the distribution characteristics between real-world datasets and artificial ones. Also the results revealed the existence of a common tendency that for real world datasets more than half of the concepts lie in the area corresponding to concepts with a small size of extent.

This observation has led to the hypothesis that for the JSM-method of hypotheses generation the top-down strategy of performing calculations is more preferable in terms of the performed computations and the number of

explored elements than the bottom-up one. The performed empirical exploration of the eight real-world two class datasets has not confirmed this hypothesis.

Then the empirical exploration of the effect of the application of two lattice structure preserving context transformations to the real-world datasets was performed. The results indicate that the attribute clarification and attribute reduction transformations are worth to apply at the initial stages of the data preparation and preprocessing.

Chapter 5

Visualizing Concept Lattices

In this chapter we will consider the questions concerning the visualization of the line diagrams of concept lattices. In Section 5.1 the general requirements to drawing line diagrams are considered. Then we consider alternative approaches to drawing line diagrams: the layered approach (Section 5.2), force-directed approach (Section 5.3) and approaches based on additive line diagrams (Section 5.4). Also in Sections 5.2 and 5.3 two algorithms based on corresponding approaches are presented. Line diagrams produced by different approaches are compared in Section 5.5. We end this chapter by the review of the related work and summary of results.

5.1 General requirements to drawing line diagrams

In the process of the application of Formal Concept Analysis to the data analysis tasks line diagrams serve as a primary mechanism for exploring and communicating the structure of data. The quality of the drawing of line diagrams determines to a large extent the effectiveness of the exploration.

It is hard to formalize what makes a good diagram due to the subjectivity of human perception and the variety of tasks for which the drawing can be used. In [34] the qualities of a good diagram are summarized as follows: “It should be transparent, easily readable and should facilitate the qualities of the data represented”.

We will consider the question of drawing the diagrams of the lattices that have sizes up to 50 nodes. Usually the diagrams of the lattices that have bigger size are of little use due to the scantiness of the screen space.

The question of drawing line diagrams of lattices can be viewed in the context of a more general question of the graph drawing [5]. But the drawing

of line diagrams of lattices does possess some special requirements that are usually not set when drawing usual graphs.

The hard requirements for drawing concept lattices are:

1. edges should be displayed as straight lines;
2. the preservation of the partial order of the lattice in the drawing (i.e. all children nodes should be drawn below their respective parents);
3. no two vertices should be located in the same point;
4. an edge should not cross the vertex that does not belong to the edge vertices.

The criteria 1 and 2 are specific to drawing Hasse diagrams.

The soft aesthetic criteria include the following ones:

- the number of edge crossings should be minimized;
- the number of different direction vectors used in the drawing should be minimized;
- the number of parallel lines should be maximized;
- it is desirable that elements of a single chain were located on one line.

5.2 The layered approach

Our first attempt to solve the lattice visualization problem was based on the modification of the approach proposed in [29], which, in a turn, is based on the widely known Sugiyama's approach for drawing hierarchical graphs [74].

The approach applied by us works as follows.

First, the nodes of a lattice are ranked. As a rank function the height of the element in the lattice is used. The Hasse diagram is considered as a directed graph (digraph). Then the digraph of the lattice Hasse diagram is converted to a proper layered graph.¹

The conversion is performed by splitting edges with vertices, the difference of rank of which exceeds one. These edges are replaced by chains of edges with new dummy nodes.

¹The digraph (G, E) , where G is a set of vertices and E is a set of edges, is called a *layered digraph*, when the set of vertices G is divided into disjoint sets L_1, \dots, L_n , $\cup_{i=1}^n L_i = G$ in such a way, that when $(u, v) \in E$ (i.e. (u, v) is an edge) and $u \in L_i$ and $v \in L_j$ then $j > i$. A *proper layered digraph* is a layered digraph such that, if $(u, v) \in E$ with $u \in L_i$ and $v \in L_j$ then $j = i + 1$.

The next step of the procedure is the minimization of the number of edge crossings. As a result of the procedure of edge crossing minimization for each layer of the digraph the order of vertices inside the layer is specified.

Finding a diagram with a minimum number of edge crossings is an NP-hard task [36]. Also it has been shown that the problem of minimizing edge crossings in a bipartite graph where one of the layers has a prescribed order is NP-Hard [23]. This result was transferred also to the case of lattices by Ralph Freese in [27]. That is why for solving the task of minimizing of edge crossings generally heuristics are employed.

The minimization of the number of edge crossings in our case is performed using the layer-by-layer sweep. First, some ordering of the elements of the second layer is selected (the first and the last layer contain only one element, namely, top and bottom elements of lattice, and that is why the ordering of these layers is trivial). After that for each next layer L_i the ordering of the vertices that minimize edge crossings is calculated while keeping the ordering of vertices of L_{i-1} fixed. The sweep operation is performed alternately top-down and then bottom-up. The operation is being performed while some improvement in reducing the total number of crossings is achieved and the number of iterations does not exceed some predetermined limit (namely, three iterations in our case).

On the next stage of the algorithm the coordinates are assigned to the lattice nodes. The y coordinate is assigned on the basis of the layer of the node. The procedure for assigning x coordinates is a more complicated one.

The coordinates of the nodes of the next level L_{i+1} are assigned on the basis of the coordinates of the current level L_i in such a way that the order calculated in the previous phase is preserved. If conflicts between the locations of several nodes arise (i.e. the distance between nodes is less than a minimal allowed distance between nodes in one layer) then they are resolved by shifting x -coordinates of nodes in a way that preserves their ordering. The procedure is performed in a sweeping move, starting from the top element of the lattice and going downwards and then starting from the bottom element of the lattice and going upwards until the coordinates of the elements stabilize or the maximal number of sweep moves is achieved.

5.3 The force-directed approach

A widely used group of algorithms for drawing general graphs are the methods based on modeling the interaction of forces, so called *force-directed methods*. The force directed algorithms are based on the simulation of a system of forces defined on an input graph and outputs a locally minimal energy

```

procedure ForceDirectedLayout
   $iterationCount = |Lattice|$ 
  for each  $i \in 1..iterationCount$ 
    for each  $Node1 \in Lattice$ 
      for each  $Node2 \in Lattice$ 
        if  $distance(Node1, Node2) \leq d_{neigh}$ 
           $f_{rep} := repulsionForce(Node1, Node2, i)$ 
           $updatePosition(f_{rep}, Node1, Node2)$ 
        for each  $Node2 \in Node1.Covered$ 
           $f_{att} := attractionForce(Node1, Node2, i)$ 
           $updatePosition(f_{att}, Node1, Node2)$ 

```

Figure 5.1: The scheme of the force-directed algorithm for concept lattice layout. Here d_{neigh} is a distance, up to which the repulsion force is taken into account. For the details of the calculations of forces see text.

configuration. The two ingredients of such an approach are a force model that specifies the forces acting between vertices and edges of the graph and a technique for finding a locally minimal energy configuration. The advantage of such methods are that they often show symmetries existing in the graph and it is easy to adapt these methods for 3D drawings.

We have employed two force-directed methods for testing their practicality for drawing diagrams of concept lattices. The first method is a simple force-directed scheme. The second method is the method proposed by Ralph Freese [27].

5.3.1 Force-directed scheme

Usually force-directed methods are applied for drawing undirected graphs. Due to the requirement of preserving the partial order of nodes when drawing Hasse diagrams, a modification should be made to the general force-directed methods.

The adaptation was performed by applying a layering approach to the assignment of the y -coordinates of the lattices and then using a force-directed approach for the determining the x coordinates.

The scheme of the algorithm is shown in Figure 5.1. In this method, the following forces are used:

The attraction force between elements. The attraction between lattice

elements is calculated on the basis of the following formula:

$$f = c_{att(i)} \cdot d(v_1, v_2).$$

Here $c_{att(i)}$ is a force adjustment variable depending on the iteration number and $d(v_1, v_2)$ is the distance between elements v_1 and v_2 .

The repulsion force between elements. The repulsion force follows the inverse square law and is equal to

$$f = \frac{c_{rep(i)}}{d(v_1, v_2)^2}.$$

The repulsion forces are calculated taking into account the influence of all other nodes of the lattice located in the neighbourhood of the node (i.e. not only comparable elements, but also incomparable ones). The attraction forces were calculated taking into account only the elements directly covered by the node.

For the search for the minimum energy state the method of simulating forces interaction was employed. The number of iterations when stimulating the forces was N , where N is the cardinality of the lattice.

5.3.2 Freese method

The method of Ralph Freese for drawing lattices also belongs to the force-directed methods. The peculiarities of this method are:

- the layout is performed in the 3D space;
- the attraction forces act between the comparable elements of the lattice;
- the repulsion forces are acting between incomparable elements of the lattice that have the same rank.

To each node of the lattice a 3D point $\langle x, y, z \rangle$ is associated. The particular form of the attraction force is

$$f = c_{att(i)} \langle x_1 - x_2, y_1 - y_2, 0 \rangle.$$

The repulsion force is

$$f = c_{att(i)} \frac{\langle x_1 - x_2, y_1 - y_2, 0 \rangle}{|x_1 - x_2|^3 + |y_1 - y_2|^3 + |z_1 - z_2|^3}.$$

Our implementation differs from the original one in the following way. When calculating the attraction forces the influence of the elements of the element filter is taken into account at each odd iteration and the influence of the elements of the element ideal is taken into account at every even iteration.² The experiments with different lattices has shown that this modification leads to more stable results compared with the original approach in which all comparable elements are taken into account when calculating the attraction forces.

The procedure of the search for the minimum energy state of the Freese method is similar to the one described in [28]. The simulation of the force interaction is performed in three distinct phases. In the first phase repulsion forces prevail, in the second one attraction forces prevail and in the third phase the forces are balanced.

5.4 Additive line diagrams

Several of the currently used approaches to drawing concept lattices are using the idea of additive line diagrams, proposed by Ganter and Wille [34]. Let the lattice $\underline{\mathcal{B}}(G, M, I)$ be given. The additive line diagram is specified with the help of two functions:

1. The representation function rep that sets each element of the lattice in correspondence to some subset of the powerset of a set X ,

$$rep : \underline{\mathcal{B}}(G, M, I) \mapsto \mathcal{P}(X).$$

The set X is called the *representation set*. The important property of the representation function is the preservation of the partial order of the lattice, i.e. from $E \leq F$ follows $rep(E) \subseteq rep(F)$, where E and F are concepts.

2. For each element x of the representation set X some real-valued, usually two-coordinate vector $vec(x)$ is set in correspondence.
3. The placement of the lattice elements is determined as follows. The coordinates of the concepts in the additive line diagram are specified with the help of the following formula:

$$coord((A, B)) = n + \sum_{x \in rep((A, B))} vec(x),$$

where n is a vector used to shift the location of the lattice on the display.

²For the definition of element ideal and filter see p. 8.

5.4.1 Embedding of the lattice into n -dimensional grid

One of the most widely used methods for drawing concept lattices is based on the following theorem. The \vee -dimension of the finite lattice L (i.e. the minimal number of chains in which the lattice L can be \vee embedded) equals the width of the set of \wedge -irreducible elements of L . Usually the set of all irreducible attributes of the context of concept lattice is used as the representation set. The decomposition of the partial order of the set of irreducible attributes is performed by means of a version of Ford-Fulkerson algorithm.³ This method is based on embedding the drawing of lattice into the grid formed by the product of chains, i.e. when there are n independent chains in the lattice then the drawing is embedded in an n -dimensional grid.

5.5 Comparison of methods

5.5.1 Test lattices

For the comparison of different drawing approaches a set of test lattices was used. It was compiled of the lattices that are either typical for the lattice theory or the lattices that were previously used in the FCA literature.

The following lattices was used:

- **B₄**: the lattice of boolean algebra consisting of 16 elements ;
- **FD₃**: the free distributive lattice with three components;
- **ID₄**: the interordinal scale [34];
- the triangles example, i.e. the lattice describing the geometric properties of triangles [34];
- the lattice describing the interrelationships between different properties of finite lattices [83];
- the lattice obtained as the result of processing the repertory grid⁴ of student's attitude to different lectures [75];
- the lattice obtained as the result of the processing of the repertory grid of an anorectic patient [12];

³The Ford-Fulkerson algorithm is an algorithm for calculating a maximal flow in a graph in which edges have a flow capacity.

⁴The repertory grid method is a knowledge elicitation technique based on the works of G. Kelly [42].

- the lattice describing relationships between different colors [85].

5.5.2 Results

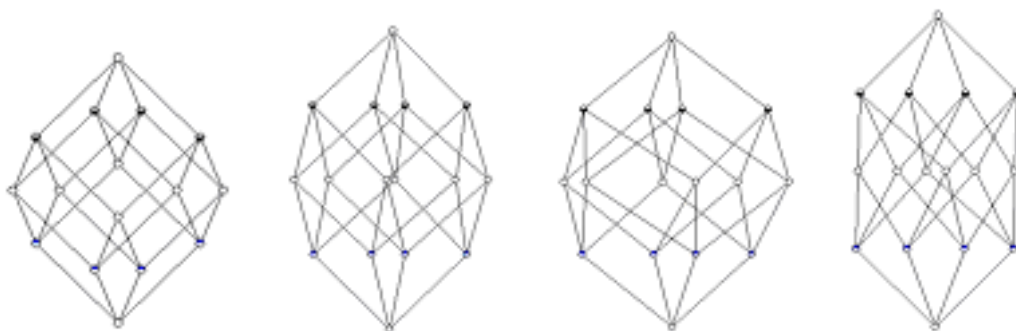
The layouts of the test lattices obtained as a result of the comparison are shown in Figures 5.2, 5.3, 5.4. From the figures it can be seen that there was no single generally best algorithm.

For distributive lattices (i.e. \mathbf{B}_4 and \mathbf{FD}_3) the method of drawing the lattice by embedding the drawing into a grid was the best one. However, for the non-distributive lattices this method suffers from the problem that is characteristic for the additive line diagrams methods using an attribute-based representation in general. The problem is that those elements of the lattice differing in significant number of attributes from their parents (for example, the bottom element of the lattice) have significantly larger than average distance from their parents in the line diagram. This effect can be observed in the layouts of the test lattices **Colors**, **ID₄**, **Repertory grid of anorectic patient** and others. The effect is caused by the fact that y -components of the direction vectors are always positive. Therefore if the element intent differs significantly from its parents' intents, then it will be located further down in the diagram.

The method of drawing line diagrams with the minimization of the number of edge crossings leads mostly to highly interpretable diagrams. Especially good are the results for lattices with planar or close to planar line diagrams or for lattices that have “almost” tree-like structure (**ID₄**, **repertory grid of anorectic patient**, **colors**). At the same time the drawings produced for lattices that possess a regular structure, such as \mathbf{B}_4 and \mathbf{FD}_3 , are worse than the ones produced by other methods. The main drawbacks of the method are its high computational time for large diagrams and absence of a symmetrical display.

The force-directed methods for drawing lattices produce nice drawings of lattices in some cases. They are good for drawing lattices with non-planar Hasse diagram, like **student repertory grid**, \mathbf{FD}_3 and **relationships of lattice properties**. They reveal symmetries in the lattices. The drawings do not satisfy criteria of maximizing the number of the parallel lines. Also, elements of a single chain are not usually located on one line. The force directed methods are characterized by high computation time. Very often the tuning of parameters in order to obtain a satisfactory diagram is required.

The drawing produced by two force-directed methods were of comparable quality in most case, with a small preference for the Freese method (compare, for example, the drawings of the **triangles properties** lattice). Also in experiments for some configurations of parameters force-directed methods

B_4 

Colors

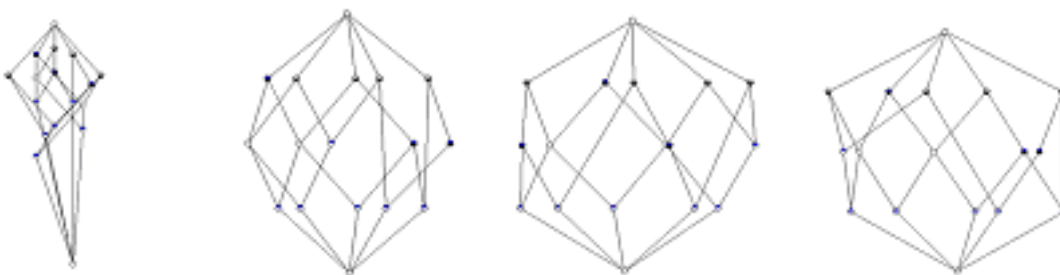
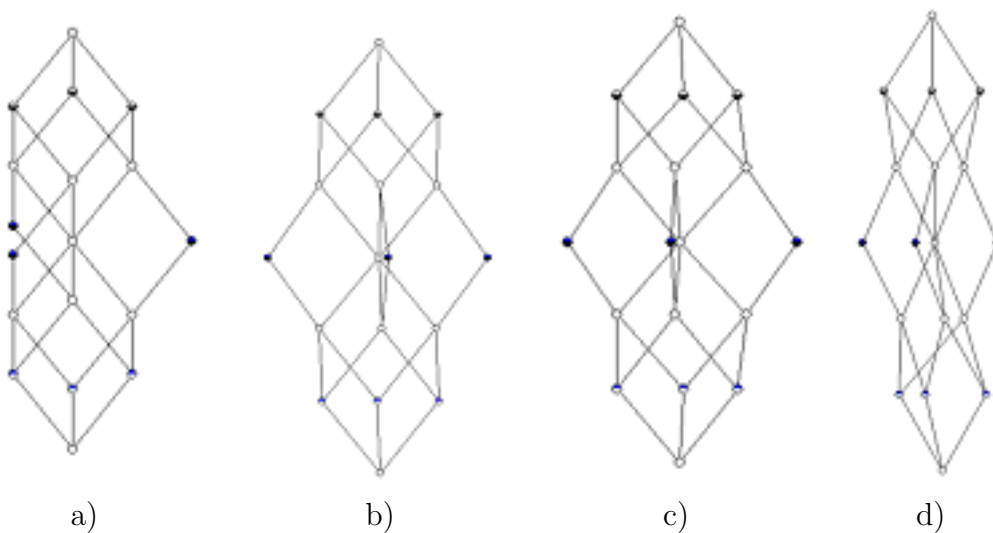
 FD_3 

Figure 5.2: The line diagrams of three test lattices produced by the methods of: a) grid embedding b) force directed layout c) Freese d) minimization of edge crossings.

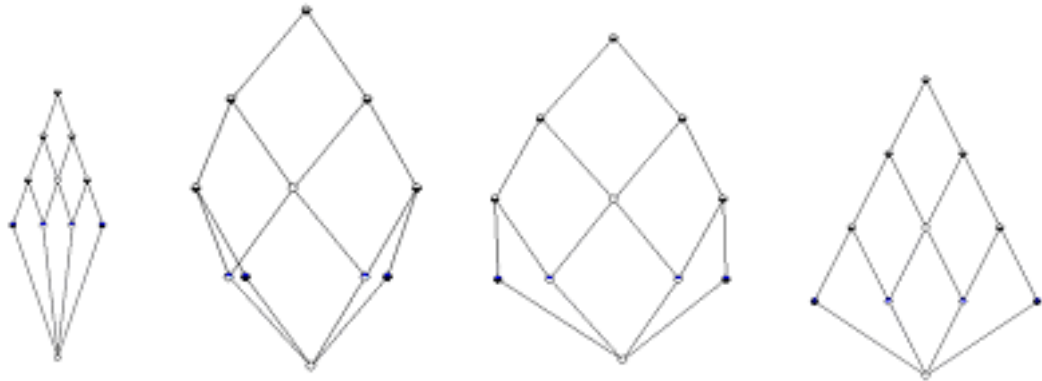
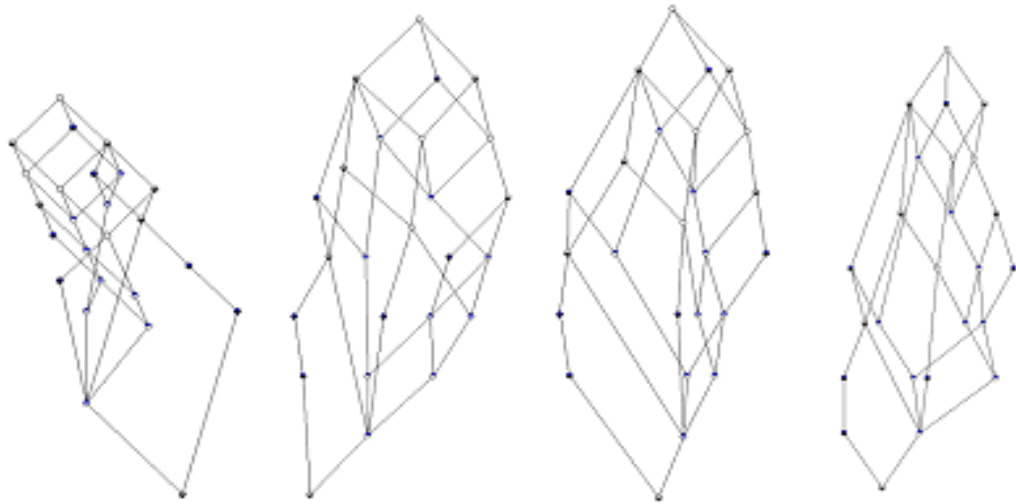
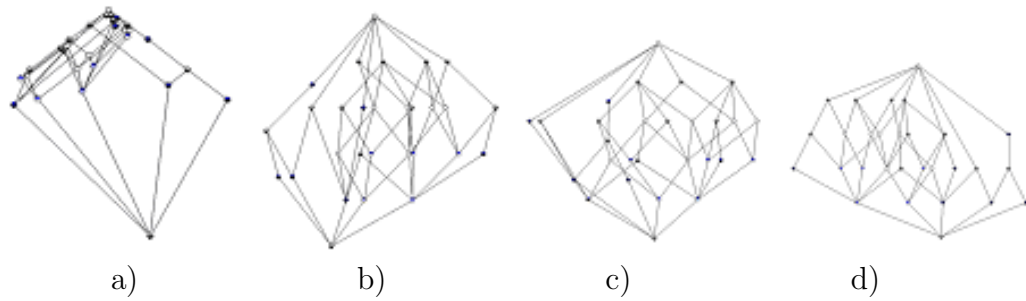
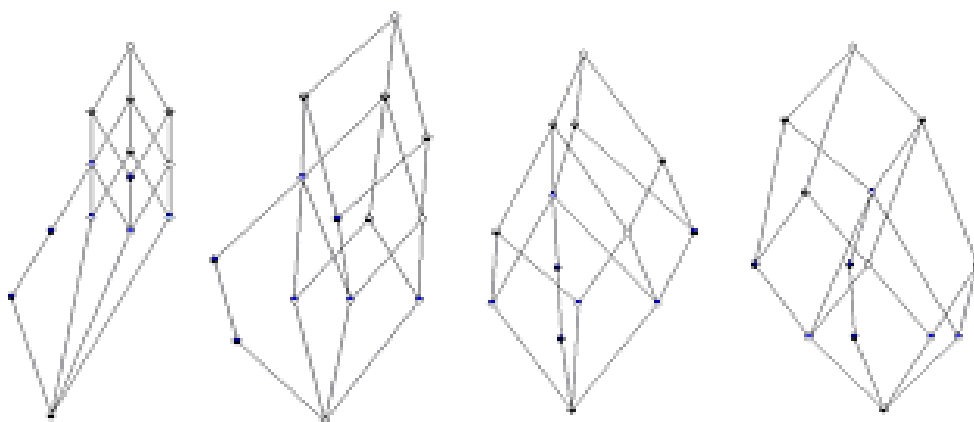
ID₄**Relationships of lattice properties****Repertory grid of anorectic patient**

Figure 5.3: The line diagrams of three test lattices produced by the methods of: a) grid embedding b) force directed layout c) Freese d) minimization of edge crossings.

Student repertory grid



Triangles properties

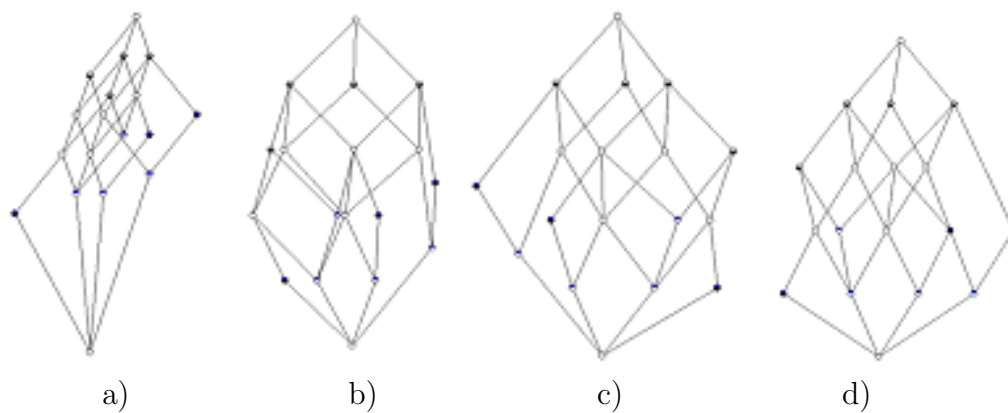


Figure 5.4: The line diagrams of the “Student repertory grid” and “Triangles” lattices, produced by the following layout methods: a) grid embedding b) force directed layout c) Freese d) minimization of edge crossings.

showed unstable behaviour.

In general, based on our experience, we would recommend to try several methods when exploring the concept lattice. As a method of first choice either the method with the minimization of the number of edge crossings or the method of embedding the lattice into a grid can be used. If the results that are produced by these methods are not quite satisfactory then the force-directed methods can be employed.

5.6 Related work

Several methods for drawing concept lattices were developed in Darmstadt in works of Wille [84], Luksch, Skorky and Wille [51], and others.

When drawing a small lattice, FCA practitioners often use so-called geometric heuristic [73]. The heuristic is based on the construction of a special auxiliary drawing (called a geometrical diagram) on the basis of the lower neighbours list of lattice elements and then using the patterns revealed when drawing such a picture for producing the drawing of the line diagram.

The rule of the parallelogram was proposed by Martin Skorsky [68]. The idea of the method is that the quadruples of lattice elements a, b, c, d , such that $a \prec \{b, c\} \prec d$, where \prec is a lattice covering relation, should be displayed as a parallelogram in the drawing of the lattice. Skorsky reports that this heuristic works good for locally-distributive lattices.

Richard Cole also considered questions of drawing concept lattices [15, 16, 17]. He combined methods of force-directed placement with additive line diagrams and also proposed a hybrid method that combines a layered approach to drawing hierarchical graphs with additive line diagrams. He used several evaluation functions in order to evaluate the goodness of the lattice and employed limited backtracking search in order to find the set of diagrams that are Pareto-optimal according to the employed evaluation functions. We have implemented his method. In our experience the method required very high computation times and due to this fact was able to work interactively only with lattices of a small size (in average, about 15 nodes).

Peter Becker [7] proposed an approach for drawing concept lattices using a multi-dimensional representation. His approach is also using additive line diagrams. The main strong point of the method is its suitability for interactive explorations. The method allows to animate the process of extension of a current diagram to a new one, when the user adds some attribute to the set of explored attributes.

Bernhard Ganter in [31] considered the question of finding a best position for the node in an additive line diagram when changing the position of the

node. This idea was further developed by B. Schmidt in [66].

5.7 Conclusions and future work

In this chapter we presented two new methods for drawing concept lattices and performed their comparison with existing methods. One of the methods, namely, the method based on the minimization of the number of edge crossings compares well with the existing methods. The drawings, produced by the other method, namely, the adaptation of force-directed algorithm, are usually a little bit less attractive, but also produce aesthetically pleasant lattice layouts in some cases.

In our practical experience the methods considered in the chapter were able to work interactively with lattices of a size up to several hundred nodes.

The algorithms considered in the chapter were implemented in JavaTM programming language and are available in the “Concept Explorer” system [85].

One direction of future work is the development of methods that would allow to perform interactive explorations of the big lattices, produced in the process of the iceberg lattice construction. Usually, such lattices can be quite big, up to several thousands or even tens or hundreds of thousands of elements. Usually, only a minor part of the elements are of special interest to the user. For exploring such lattices from our point of view other methods of layout would be required, that do not necessarily preserve the structural information of the lattice, but pay more attention to the quantitative information of the concept part.

An example of such a method can be a tool in which the vertical coordinate of the concept node is assigned based on the cardinality of the concept extent. Such an assignment preserves the partial order of the lattice. Other desirable properties of a tool would be the special navigation interface that allows to explore a selected part of the big concept lattice (possibly similar to the fish-eye method [65], frequently employed for the exploration of big graphs) and also allows to perform a local layout of the selected parts of the concept lattice.

Chapter 6

Conclusions

In this chapter we highlight the main contributions of our research and summarize the main results. Then, we present some potential avenues for future research.

6.1 Contributions

In this thesis we studied the algorithmic aspects of the Formal Concept Analysis and related methods of data mining and knowledge discovery in databases. Our contribution to the field includes the development of new algorithms for the solution of the tasks of computing the set of all concepts and the corresponding Hasse diagram, an extensive search space analysis of the characteristics of some real-world datasets, and the development of new methods for the visualization of concept lattices.

The main contributions are highlighted as follows.

Algorithms for the calculation of the set of all concepts (Chapter 3)

We have presented a novel algorithm for the calculation of the set of all concepts and family of algorithms for the calculation of the set of all intents and pointed out improvements to one of the best existing algorithms.

The first algorithm can serve as the algorithmic basis for solving different tasks in Formal Concept Analysis. It shows very good performance in practice and has low memory consumption.

The family of algorithm for calculating the set of all intents is based on the usage of implicit representations (namely, Binary Decision Diagrams) and shows excellent performance for examples with dense contexts.

An experimental comparison of these two algorithms with a number of existing algorithms was performed. It has shown that there is no single generally best algorithm that works equally well for all classes of contexts.

Search space analysis (Chapter 4)

An extensive exploration of the search space characteristics of data mining datasets typical for association rule mining was performed.

A significant difference in the distribution characteristics between real-world datasets and artificial ones was revealed. Also the results revealed the existence of a common tendency that for real world datasets more than a half of the concepts lie in the area corresponding to concepts with a small size of extent. This result shows the way of improving the performance of algorithms by creating their special versions for computing concepts with small extent size.

An empirical exploration of the effect of the application of lattice structure preserving context transformations on real-world datasets was performed. The results indicate that the attribute clarification and attribute reduction transformations are worth to apply at the initial stages of the data preparation and preprocessing.

Visualizing concept lattices (Chapter 5)

We have presented two new methods for drawing concept lattices: an algorithm for drawing concept lattices that minimizes the number of edge crossings and a force-directed algorithm for drawing concept lattices. Their comparison with existing methods showed that the first approach is a strong competitor with other algorithms and often produces aesthetically pleasant drawings.

All of the algorithms described in the thesis were implemented in the research software tools “Concept Explorer” (ConExp) [85] and “QuDA” [41], that are available online at <http://www.sf.net/projects/conexp/> and <http://kirk.intellektik.informatik.tu-darmstadt.de/~quda/>. The ConExp software enjoyed popularity among users and its last version was downloaded more than 500 times from the date of its publication at 28 of July 2003.

6.2 Future work

From our viewpoint, the most interesting open issues in the area of our research are:

- Research oriented towards improving the theoretical estimates of the asymptotic complexity of algorithms used in Formal Concept Analysis and establishing better lower bounds on the complexity of algorithms or proving that estimates are the optimal one.
- An exploration of the attribute ordering strategies for the algorithms for computing the set of all intents using Binary Decision Diagrams.
- The development of algorithms using implicit representations for solving other tasks, such as the calculation of the set of all generators of context or of the Duquenne-Guigues basis.
- A comprehensive experimental comparison of the set of algorithms used in Formal Concept Analysis, that were developed by different groups in order to gather the best practical experiences in the area of implementation techniques and create a publicly available base of the implementations and experimental contexts.
- The development of techniques allowing an interactive exploration of big lattices, up to several thousand nodes or more, which are also suited for the display of quantitative information contained in the lattices.

Bibliography

- [1] Rakesh Agrawal, Tomasz Imielinsky, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 International Conference on Management of Data(SIGMOD 93)*, pages 207–216, 1993.
- [2] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, 1978.
- [3] H. R. Anderson. An introduction to binary decision diagrams, 1998. <http://www.itu.dk/people/hra/bdd97.ps>.
- [4] Yves Bastide, Nicolas Pasquier, Rafik Taouil, Gerd Stumme, and Lotfi Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L.M. Pereira, Y. Sagiv, and P.J. Stuckey, editors, *Computational Logic – CL 2000. Proceedings of CL 00.*, number 1861 in LNAI, pages 972–986, Heidelberg, 2000. Springer.
- [5] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing*. Prentice-Hall, 1999.
- [6] Stephen D. Bay and Michael J. Pazzani. Detecting Change in Categorical Data: Mining Contrast Sets. In *Knowledge Discovery and Data Mining*, pages 302–306, 1999.
- [7] Peter Becker. Multi-dimensional representations of conceptual hierarchies. In Gerd Stumme and Guy Mineau, editors, *Proceedings of the 9th International Conference on Conceptual Structures. Supplementary Proceedings ICCS*, pages 33–46, Department of Computer Science, University Laval, 2001.
- [8] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. <http://www.ics.uci.edu/~mlearn/MLRepository.html>.

- [9] J.P. Bordat. Calcul pratique du treillis de galois d'une correspondance. *Math. Sci. Hum.*, (96):31–47, 1986.
- [10] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.
- [11] Douglas Burdick, Manuel Calimlim, and Johannes Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *Proceedings of the 17th International Conference on Data Engineering*, pages 443–452, 2001.
- [12] Peter Burmeister. Formal concept analysis with ConImp: Introduction to the basic features. <http://www.mathematik.tu-darmstadt.de/~burmeister/ConImpIntro.pdf>.
- [13] Claudio Carpineto and Giovanni Romano. A lattice conceptual clustering system and its application to browsing retrieval. *Machine Learning*, (24):95–122, 1996.
- [14] M Chein. Algorithme de recherche des sous-matrices premières d'une matrice. *Bull. Math. Soc. Sci. Math. R. S. Roumanie*, (13):21–25, 1969.
- [15] Richard Jeffrey Cole. Automatic layout of concept lattices using force directed placement and genetic algorithms. In Jenny Edwards, editor, *23th Australasian Computer Science Conference*, pages 47–53. IEEE Computer Society, 2000.
- [16] Richard Jeffrey Cole. *The management and visualization of Document Collections using Formal Concept Analysis*. PhD thesis, Griffith University, 2000.
- [17] Richard Jeffrey Cole. Automated layout of concept lattices using layer diagrams and additive diagrams. In Michael Oudshoorn, editor, *24th Australasian Computer Science Conference, IEEE Computer Society*, pages 47–53, 2001.
- [18] Laurentiu Cristofor. ARMiner software. <http://www.cs.umb.edu/~laur/ARMiner/>.
- [19] Laurentiu Cristofor. *Mining rules in single-table and multiple-table databases*. PhD thesis, University of Massachusetts, Boston, June 2002.
- [20] Jean-Paul Doignon and Jean-Claude Falmagne. *Knowledge Spaces*. Springer, 1999.

- [21] Cornelia E. Dowling. On the irredundant construction of knowledge spaces. *Journal of Mathematical Psychology*, 37:49–62, 1993.
- [22] V. Duquenne and J.-L. Guigues. Familles minimales d’implications informatives resultant d’un tableau de donnees binaires. *Math. Sci. Humaines*, 95:5–18, 1986.
- [23] P. Eades and N. Wormald. Edge crossing in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994.
- [24] V.K. Finn. On a generalized method of automatic hypothesis generation. *Semiotic and informatic*, 29, 1989. In Russian.
- [25] V.K. Finn. Plausible reasoning in intelligent systems of JSM-type. *Itogi Nauki i Tekhniki, ser. Informatika*, (15):54–101, 1991. In Russian.
- [26] Van Der Merwe F.J. and Kourie D.G. AddAtom: an incremental algorithm for constructing concept lattices and concept sublattices. Technical report, Department of Computer Science, University of Pretoria, 2002.
- [27] Ralph Freese. Automated lattice drawing. In Peter Eklund, editor, *Concept Lattices. Second International Conference on Formal Concept Analysis. Proceedings*, volume 2961 of *LNAI*, pages 112–123, 2004.
- [28] Thomas Fruchterman and Edward Reingold. Graph drawing by force directed placement. *Softw. - Pract. Exp.*, 21(11):1129–1164, 1991.
- [29] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. A technique for drawing directed graphs. *Software Engineering*, 19(3):214–230, 1993.
- [30] B. Ganter and S.O. Kuznetsov. Formalizing hypotheses with concepts. In G. Mineau and B. Ganter, editors, *Proc. 8th Int. Conf. on Conceptual Structures, ICCS’00*, volume 1867 of *Lecture Notes in Artificial Intelligence*, pages 342–356. Springer-Verlag, 2000.
- [31] Bernhard Ganter. Conflict avoidance in additive order diagrams. <http://www.math.tu-dresden.de/~ganter/psfiles/additive.ps>.
- [32] Bernhard Ganter. Two basic algorithms in concept analysis. *FB4-Preprint No 831*, 1984.

- [33] Bernhard Ganter. Formal Concept Analysis: Algorithmic Aspects, 2002. Lecture notes for course "Computational Logic: Advanced Unit: Formal Concept Analysis".
- [34] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical foundations*. Springer-Verlag, Berlin Heidelberg New-York, 1999.
- [35] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, San Francisco, CA, 1979.
- [36] M. R. Garey and D. S. Johnson. Crossing number is np-complete. *SIAM J. Algebraic Discrete Methods*, 4(3):312–316, 1983.
- [37] Robert Godin, Rokia Missaoui, and Hassan Alaoui. Incremental concept formation algorithms based on galois lattices. *Computation Intelligence*, 11(2):246–267, 1995.
- [38] Bart Goethals. *Efficient Frequent Pattern Mining*. PhD thesis, Transnationale Universiteit Limburg, School voor Informatietechnologie, 2002.
- [39] Gošta Grahne and Jianfei Zhu. Efficiently using prefix-trees in mining frequent itemsets. <http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-90/grahne.pdf>.
- [40] Peter Grigoriev. *Methods of intelligent data analysis in domains with partially determined properties of objects*. PhD thesis, Russian Humane University, 2000.
- [41] Peter Grigoriev, Serhiy Yevtushenko, and Gunter Grieser. QuDA, a data miner's discovery environment. Technical Report AIDA-03-06, FG Intellektik, FB Informatik, Technische Universität Darmstadt, September 2003. <http://www.intellektik.informatik.tu-darmstadt.de/~peter/QuDA.pdf>.
- [42] G. A. Kelly. *The Psychology of Personal Constructs*. W.W. Norton, 1955.
- [43] Willi Kloesgen. Explora: A multipattern and multistrategy discovery assistant. In Usama M. Fayad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 249–271. MIT Press, 1996.

- [44] Sergei O. Kuznetsov. The JSM-method as a system of machine learning. *Itogi Nauki i Tekhniki, ser. Informatika*, (15):17–54, 1991. In Russian.
- [45] Sergei O. Kuznetsov. A fast algorithm for computing all intersections of objects in a finite semi-lattice. *Automatic Documentation and Mathematical Linguistics*, 24(5):11–21, 1993.
- [46] Sergei O. Kuznetsov. *Theory of Machine Learning in Formal Concept Lattices*. PhD thesis, VINITI, Russian Academy of Science, 2002. In Russian.
- [47] Sergei O. Kuznetsov and Sergei A. Ob’edkov. Comparing performance of algorithms for generating concept lattices. In *Proc. of ICCS’01 - Int’l Workshop On Concept Lattices Based KDD*, pages 35–47, 2001.
- [48] Christian Lindig. Concept based component retrieval. In Jana Köhler, Fausto Gianchiglia, Cordell Green, and Cristoph Walther, editors, *Working Notes of the IJCAJ-95 Workshop: Formal Approaches to the Reuse of plans, Proofs, and Programs*, pages 21–25, Montreal, August 1995.
- [49] Christian Lindig. *Algorithmen zur Begriffsanalyse und ihre Anwendungen bei Softwarebibliotheken*. PhD thesis, Technische Universität Braunschweig, 1999.
- [50] Christian Lindig. Fast concept analysis. In Gerd Stumme, editor, *Working with Conceptual Structures: Contributions to ICCS 2000*, pages 152–161, 2000.
- [51] Peter Luksch, Martin Skorsky, and Rudolf Wille. On drawing concept lattices with computer. In W. Gaul and M. Schader, editors, *Classification as a tool of research*, pages 269–274. North Holland, Amsterdam, 1986.
- [52] Michael Luxenberger. *Implikationen, Abhängigkeiten und Galois Abbildungen*. PhD thesis, Technische Hochschule Darmstadt, 1993.
- [53] S. Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *Proc. of DAC 93*, pages 272–277, 1993.
- [54] Daniele Nardi and Ronald J. Brachmann. *The Description Logic Handbook*, chapter An Introduction to Description Logics. Cambridge University Press, 2001.

- [55] E. M. Norris. An algorithm for computing the maximal rectangles in a binary relation. *Revue Roumanie de Mathématiques Pures et Appliquées*, (23(2)):243–250, 1978.
- [56] L. Nourine and O. Raynaud. A fast algorithm for building lattices. *Information Processing Letters*, 71:199–204, 1999.
- [57] Sergei Obiedkov. *Algorithms and methods of lattice theory and their application in machine learning (In Russian)*. PhD thesis, Russian State Humane University, 2003.
- [58] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Pruning closed itemset lattices for association rules. In *Proceedings of the BDA French Conference on Advanced Databases, October 1998.*, pages 177–196, 1998.
- [59] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings of the 7th ICDT Conference*, pages 398–416, January 1999.
- [60] Jian Pei, Jiawei Han, and Runying Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30, 2000.
- [61] Jian Pei, Jiawei Han, and Runying Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30, 2000.
- [62] PISSARO system – Picturing Interactively Statistically Sensible Association Rules Reporting Overviews. <http://stats.math.uni-augsburg.de/Pissarro/>.
- [63] Susanne Prediger. Terminologische Merkmalslogik in der Formalen Begriffsanalyse. In Gerd Stumme and Rudolf Wille, editors, *Begriffliche Wissensverarbeitung. Methoden und Anwendungen*, pages 99–124. Springer, Berlin-Heidelberg-New York, 2000.
- [64] Quest synthetic data generation code. <http://www.almaden.ibm.com/software/quest/Resources/datasets/syndata.html>.
- [65] Manojit Sarkar and Marc H. Brown. Graphical fisheye views. *Communications of the ACM*, 37(12):73–84, 1994.

- [66] Bernhard Schmidt and Christian Zschalig. Drawing concept lattices. In *Using Conceptual Structures. Contributions to ICCS 2003.*, pages 277 – 279, 2003.
- [67] Michael Siff and Thomas Reps. Identifying modules via concept analysis. In *Proc. of the International Conference on the Software Maintenance*, pages 170–179. IEEE Computer Society Press, October 1997.
- [68] Martin Skorsky. *Endliche Verbände – Diagramme und Eigenschaften*. PhD thesis, TH Darmstadt, 1992.
- [69] Gregor Snelting and Frank Tipp. Reengineering class hierarchies using concept analysis. In *Proc. SIGSOFT Symposium on Foundations of Software Engineering*. ACM, 1998.
- [70] Ramakrishnan Srikant and Rakesh Agrawal. Mining generalized association rules. In *Proceedings of the 21th VLDB Conference*, pages 407–419, 1995.
- [71] Gerd Stumme, Rafik Taouil, Yves Bastide, Nicolas Pasquier, and Lotfi Lakhal. Fast computation of concept lattices using data mining techniques. In *Proc. 7th Intl. Workshop on Knowledge Representation Meets Databases*, 2000.
- [72] Gerd Stumme, Rafik Taouil, Yves Bastide, Nicolas Pasquier, and Lotfi Lakhal. Computing iceberg concept lattices with titanic. *Data Knowledge Engineering*, 42(2):189–222, 2002.
- [73] Gerd Stumme and Rudolf Wille. A geometrical heuristic for drawing concept lattices. In R. Tamassia and I.G. Tollis, editors, *Graph Drawing*, pages 85–98. Springer, 1993.
- [74] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, February 1981.
- [75] T. A. Taran and A. Y. Tkachev. Application of formal concept analysis in humane studies. In Bernhard Ganter and Aldo de Moor, editors, *Using Conceptual Structures. Contributions to ICCS 2003*, 2003.
- [76] Petko Valtchev, Mohamed Rouane Hacene, and Rokia Missaoui. A generic scheme for the design of efficient on-line algorithms for lattices. In Bernhard Ganter, editor, *Proceedings of 11th International Conference on Conceptual Structures, ICCS 2003*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2003.

- [77] Petko Valtchev and Rokia Missaoui. Building concept (galois) lattices from parts: Generalizing the incremental methods. In *Proceedings of 9th International Conference on Conceptual Structures, ICCS 2001*, volume 2120 of *Lecture Notes in Artificial Intelligence*, pages 290–303, 2001.
- [78] Frank Vogt and Rudolf Wille. Toscana - a graphical tool for analysing and exploring data. In Roberto Tamassia and Ioannis G. Tollis, editors, *Graph drawing*, volume 894 of *Lecture Notes in Computer Science*, pages 226–233. Springer, 1994.
- [79] Jianyong Wang, Jiawei Han, and Jian Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, Washington, DC, USA, 2003.
- [80] G. I. Webb, S. Butler, and D. Newlands. On detecting differences between groups. In *Proceedings of The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pages 256–265. AAAI Press, 2003.
- [81] Geoff I. Webb and S. Zhang. Beyond association rules: Generalized rule discovery. Submitted for publication, 2003.
- [82] Rudolf Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. In Ivan Rival, editor, *Ordered sets*, pages 445–470. Reidel, Dordrecht-Boston, 1982.
- [83] Rudolf Wille. Knowledge acquisition by methods of formal concept analysis. In E. Diday, editor, *Data Analysis, learning symbolic and numeric knowledge*. Nova Science Publisher, New York-Budapest, 1989.
- [84] Rudolf Wille. Lattices in data analysis: how to draw them with computer. In Ivan Rival, editor, *Algorithm and order*, pages 33–58. Kluwer, Dordrecht-Boston, 1989.
- [85] Serhiy A. Yevtushenko. Concept Explorer: a system for data analysis. In *Proceedings of the 7th national conference on Artificial Intelligence KII-2000*, pages 127–134, Russia, 2000. (In Russian).
- [86] Serhiy A. Yevtushenko. Research and development of algorithms for the concept lattices construction (in Russian). Master's thesis, National Technical University of Ukraine “KPI”, 2000.

- [87] Serhiy A. Yevtushenko. BDD-based algorithms for the construction of the set of all concepts. In Galia Angelova, Dan Corbett, and Uta Priss, editors, *Foundations and Applications of Conceptual Structures. Contributions to ICCS 2002*, pages 61–73, 2002.
- [88] M.I. Zabegailo, V.G. Ivashko, S.O. Kuznetsov, M.A. Micheenkova, K.P. Chazanovsky, and O.M. Anshakov. Algorithmic and software tools for the JSM method of automatic hypothesis formation (in Russian). *NTI, series 2*, 10:1–14, 1987.
- [89] Mohammed J. Zaki and Ching-Jui Hsiao. Charm: an efficient algorithm for closed association rule mining. Technical report, Rensselaer Polytechnic Institute, 1999.
- [90] Mohammed J. Zaki and Ching-Jui Hsiao. Charm: An efficient algorithm for closed itemset mining. In *2nd SIAM International Conference on Data Mining*, Arlington, 2002.
- [91] Zijian Zheng, Ron Kohavi, and Llew Mason. Real world performance of association rule algorithms. In *KDD 2001*, pages 401–406, 2001.

Serhiy Yevtushenko

Curriculum Vitae

1977, February 12	Born in Kyiv, Ukraine, in family of Anatoliy and Galina Yevtushenko
1984-1994	Study at school-gymnasium № 117 named by Lesya Ukrainka (Kyiv). Graduated with Gold Medal.
1994-2000	Study at National Technical University of Ukraine “KPI” at Applied Mathematics Department
1998, June	Bachelor degree in applied mathematics with distinction
1998-2000	Study at National Technical University of Ukraine “KPI” at Management and Marketing Department.
2000, June	Master degree in applied mathematics with distinction
	Diploma in Marketing
2001, September – 2004, April	Work as scientific worker at Intellectics group, Informatics, Technical University of Darmstadt